Essential Problem Solving Strategies for Programming– Table of Contents

A Detailed Description of Polya's Four steps of Problem Solving

1. Understand the Problem (Define the Problem)

- □ ***Carefully read*** the problem ***several times***.
- □ ***Identify*** what you are being asked to ***find***.
- □ ***Ensure*** that you ***understand all terminology***.
- □ ***Highlight*** all ***given information***.
- □ ***Identify*** all the ***information*** that ***is required*** to solve the problem.
- □ ***Identify*** the ***given information*** that ***is required*** to solve the problem.
- □ ***Identify*** any ***extraneous information*** (information that is not needed).
- □ ***Identify*** any ***missing information***.
- □ ***Do research*** to ***find*** or ***estimate*** any ***missing information***.
- □ ***Keep*** an ***open mind***.
- □ ***Do not make*** any ***unnecessary*** or ***incorrect assumptions***.
- □ ***Think logically*** and ***creatively!***
- □ ***Consult colleagues, peers, experts,*** etc.
- □ ***Do not worry*** about ***possible strategies yet***.
- □ ***Predict*** what a ***reasonable answer*** or ***range of answers*** would be.

2. Choose a Strategy

- □ ***Unleash your creative powers! Be imaginative!***
- □ ***Do not be afraid to take risks!***
- □ ***Do not dismiss any ideas*** at this stage. Feel free to be ***whacky!***
- □ ***Avoid*** feelings of ***frustration*** or ***inadequacy***.
- □ ***Do not give up quickly!***
- □ If you have the desire to quit, ***take a break*** and ***try solving the problem later***.
- □ ***Do not be afraid*** to be ***unconventional***. Perhaps you are correct and everyone else is wrong!
- □ ***Draw*** a ***diagram*** or ***visualize***.
- □ ***Compare*** the problem to an ***equivalent*** or ***similar problem*** that you have already solved.
- □ ***Compare*** the problem to a ***simpler*** but ***related problem***.
- □ ***Solve*** a ***specific example*** of the problem.
- □ ***Look for patterns***.
- □ ***Write*** a list of ***as many possible strategies*** as you can.
- □ ***Do research*** to discover if ***anyone else*** has solved the problem.

3. Carry out the Strategy

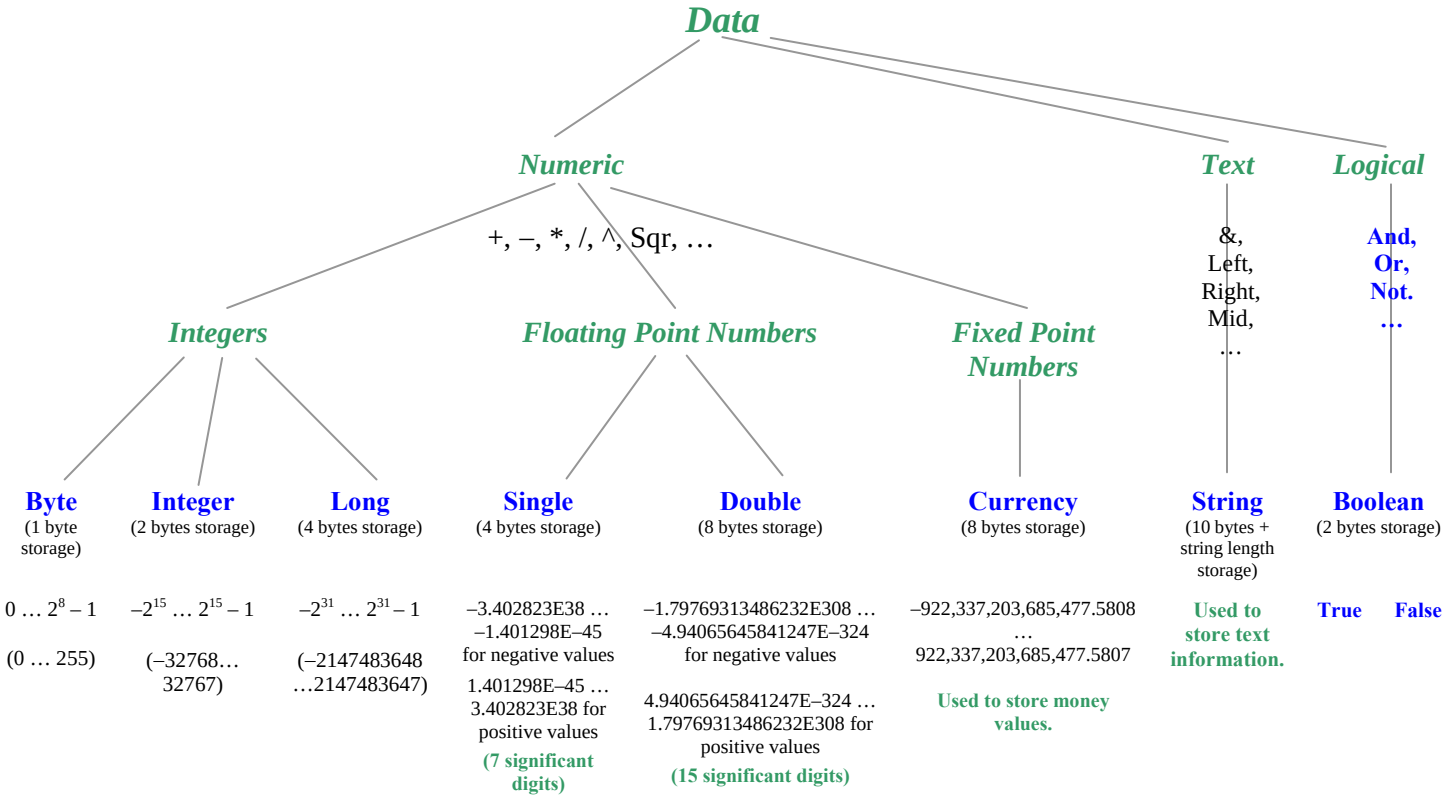
- □ ***Check*** your list of strategies and ***select one*** that you think is likely to work.
- □ ***Carry out*** your strategy ***logically*** and ***carefully***, paying close attention to ***detail***.
- □ If your strategy ***fails***, return to ***steps 1*** and ***2***.

4. Check the Solution

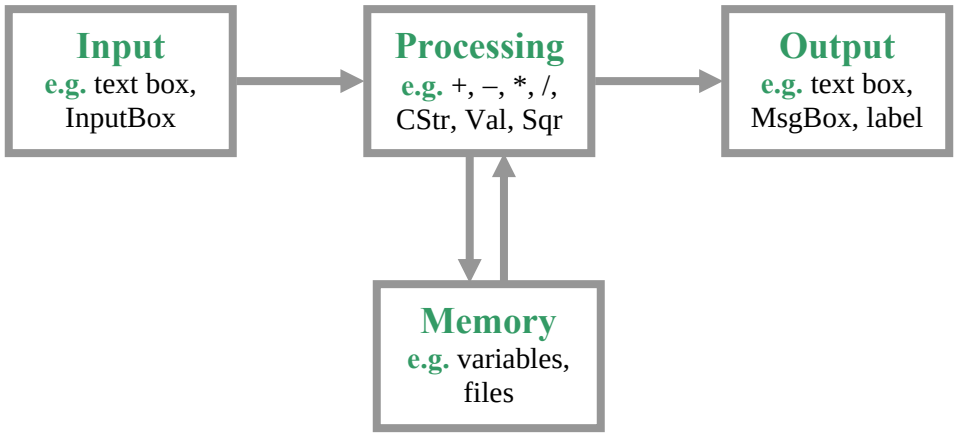
- □ Is your answer ***reasonable?***
- □ Does your ***answer agree*** with the ***prediction*** you made in ***step 1***?
- □ Does your ***answer agree*** with the ***answers obtained by others***?
- □ Is there a ***better way*** to solve the problem?
- □ Ask ***peers, colleagues,*** etc to check your solution.

Data (Information) – A Partial List of VB Data Types

A computer can be viewed as a ***data processing machine***. Since data can be categorized into various forms that require ***differing amounts of memory*** and ***different types of operations***, programming languages offer diverse ***data types***. A summary of the ***most commonly used types of data*** studied in this course is given in the following diagram.



A Computer as a Data Processing Machine



Some Useful Intrinsic (Built-In) Functions

- **Val** Converts a string value to a numeric value **e.g.** Val ("23.47") → 23.47
- **CStr** Converts any value to a string value **e.g.** CStr (23.47) → "23.47"
- **Sqr** Returns the square root of any non-negative numeric value **e.g.** Sqr (100) → 10
- **Chr** Converts an ASCII (ANSI) value to its corresponding character **e.g.** Chr (122) → "z"
- **Asc** Returns the ASCII (ANSI) value of a character **e.g.** Asc ("z") → 122
- **Trim** Remove all leading and trailing blank spaces from a string **e.g.** Trim(" Ashley Walsh ") → "Ashley Walsh"

Important Points about Data Types

- Although computer circuits can process only the binary values 0 and 1, programs need to process a wide variety of types of data including ***numbers***, ***text*** and ***logical values*** (i.e. values that are either true or false).
- ***Encoding schemes*** are used to give a ***meaning*** to raw binary data. That is, encoding schemes use binary numbers to represent information. See the table below for a few common examples of encoding schemes.
- Variables need to be declared so that both of the following are known:
  Amount of Memory Required
  Encoding Scheme that should be used to interpret the Raw Binary Data

Bits and Bytes
1 bit = 1 ***bi***nary digi***t***
1 Byte = 8 bits (1 B = 8 b)

The following table gives several examples of commonly used encoding schemes.

Type of Data	*Name of Encoding Scheme*	*Memory Required*	*Examples*	
			Raw Binary Data Stored in RAM	*What the Raw Binary Data Represent*
Integer (**Integer** in VB)	16-bit Twos Complement	2 bytes	0111111111111111	32767
String (Text)	Unicode	2 bytes	0111111111111111	翔
Integer (**Long** in VB)	32-bit Twos Complement	4 bytes	11000011100110001101000000000000	–1013395456
Floating Point (**Single** in VB)	32-bit IEEE754	4 bytes	11000011100110001101000000000000	–305.625

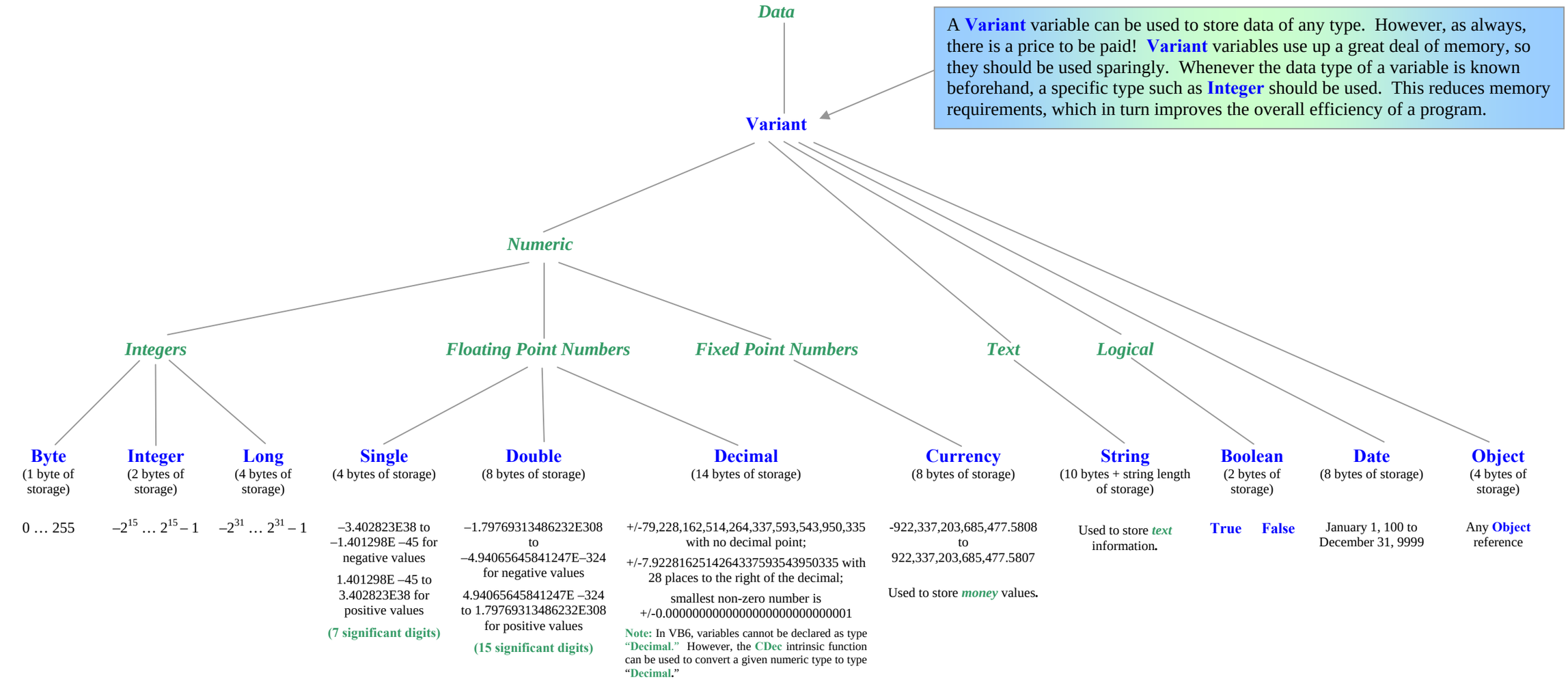
Questions

1. Why do programming languages offer so many different data types?
2. Visit www.unicode.org and find the Unicode hexadecimal (base 16) code for each of the following characters. Then use a Web-based converter or the Windows calculator to convert to binary. (Windows calculator must be in "Scientific" view.)
   (a) ち (Hiragana, Japanese) Hex code: Binary code:
   (b) ચ (Gujarati, Indic) Hex code: Binary code:
3. Now interpret the codes that you found in question 2 as 16-bit integers. Convert each code from binary form to decimal form. Again, you may use a Web-based converter or the Windows calculator.
4. Without an encoding scheme, does raw binary data have any meaning?
5. Complete the following table:

Standard Form	*Scientific Notation*	*Scientific Notation (Programming Format)*
23400000	2.34×10^{7}	2.34E7
	9.10938188×10^{-31} kg (mass of an electron)	
	1.99×10^{30} kg (mass of sun)	
		1.79769313486232E308 (largest **Double** value in VB)
0.000000475 m (wavelength of blue light)		
0.000000045 m (distance between conductors in a CPU, known as the ***fabrication process size***)		

A Complete List of Visual Basic Data Types

A computer can be viewed as a ***data processing machine***. Since data can be categorized into various forms that require differing amounts of memory, designers of programming languages separate data into diverse ***types***. A complete list of all the types of data available in VB is given in the following diagram.



A VB Program that Processes Numeric Information

Introduction

In the first unit of this course we focused entirely on programs that generate artistic designs using lines, circles and other shapes. Although these programs produced a dazzling output, they did not process a wide variety of data. Now we shall begin examining how we can use VB to create programs that process all sorts of different kinds of data. The first example deals with the processing of ***numeric data***.

Simple Addition Calculator Version 1.0

The following is a portion of the code for the "Simple Addition Calculator Version 1.0" program. You can find the complete program in the folder **I:\Out\Nolfi\Ics3mo\Simple VB Examples\Addition Calculator**. Study the program and the following notes. Then complete the questions at the end of this section.

```
Option Explicit

Private Sub cmdAdd_Click()
    'MEMORY: Declare variables to allocate memory.
    Dim Number1 As Double, Number2 As Double, Sum As Double

    'INPUT: Obtain values from user and store in variables.
    Number1 = Val(txtNumber1.Text)
    Number2 = Val(txtNumber2.Text)

    'PROCESSING: Calculate the sum and assign to "Sum"
    Sum = Number1 + Number2

    'OUTPUT: Display results.
    txtSum.Text = CStr(Sum)

    txtNumber1.SetFocus
End Sub

Private Sub cmdClear_Click()
    'Clear all the text boxes by assigning the NULL STRING
    '(empty string) to the "Text" property of each text box.
    txtNumber1.Text = ""
    txtNumber2.Text = ""
    txtSum.Text = ""

    txtNumber1.SetFocus
End Sub
```

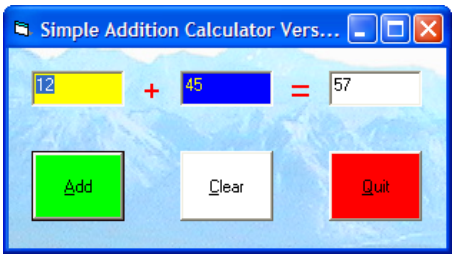
Force Variable Declarations

The numeric variables are declared as type "**Double**" to allow both whole and "non-whole" numbers to be processed.

These are ***assignment statements*** that give values to the numeric variables "Number1" and "Number2." The values are read from text boxes in "**String**" form, converted to numeric form by using the "Val" function and then stored in "Double" form using the variable names "Number1" and "Number2."

The values of the variables "Number1" and "Number2" are recalled from RAM (main memory) and added. The result is stored in RAM using the variable name "Sum."

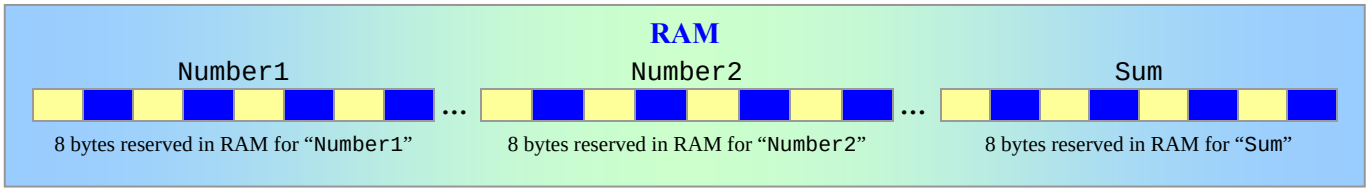
The value of the variable "Sum" is recalled from RAM (main memory). It is converted from ***numeric form*** (**Double**) to ***text form*** (**String**) and then assigned to the "Text" property of the "txtSum" text box. The "Text" property is itself a variable, which means that its value is stored in RAM.



A Pictorial Description of the Addition Calculator Program

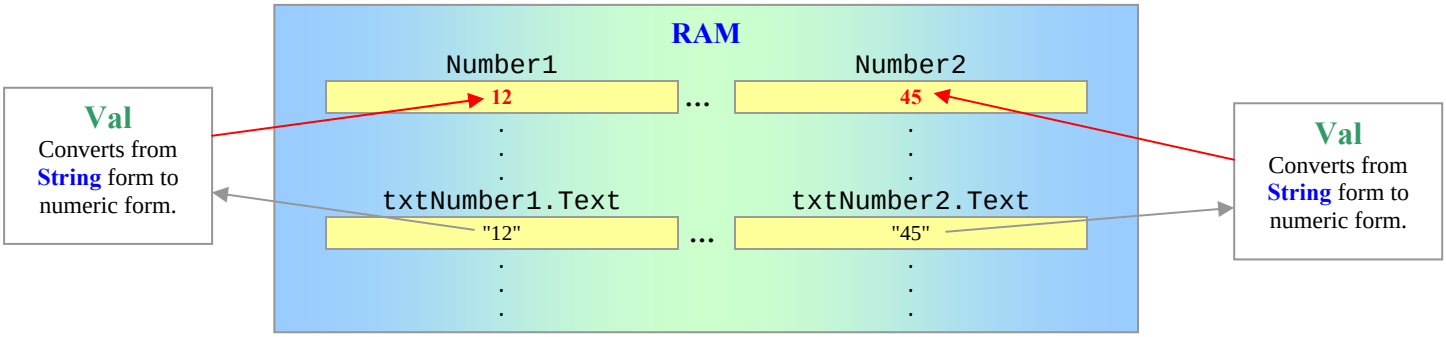
1. The first statement in the "cmdAdd_click" sub is called a variable declaration. It is used to state the name and type of variables. The diagram below shows the effect of this statement on RAM (main memory).

```
'MEMORY: Declare variables to allocate memory.
Dim Number1 As Double, Number2 As Double, Sum As Double
```



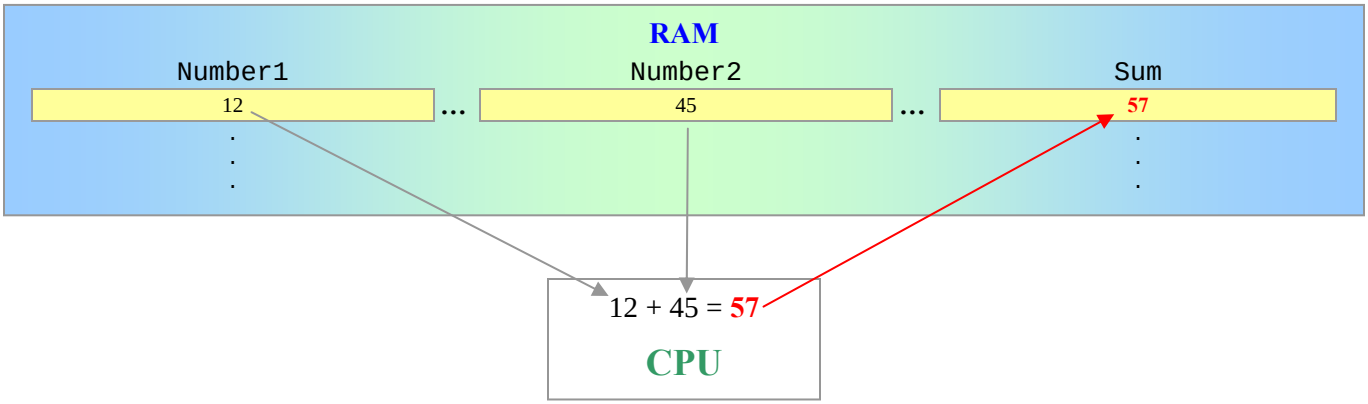
2. The next two statements are used to get ***input*** from the user.

```
'INPUT: Obtain values from user and store in variables.
Number1 = Val(txtNumber1.Text)
Number2 = Val(txtNumber2.Text)
```



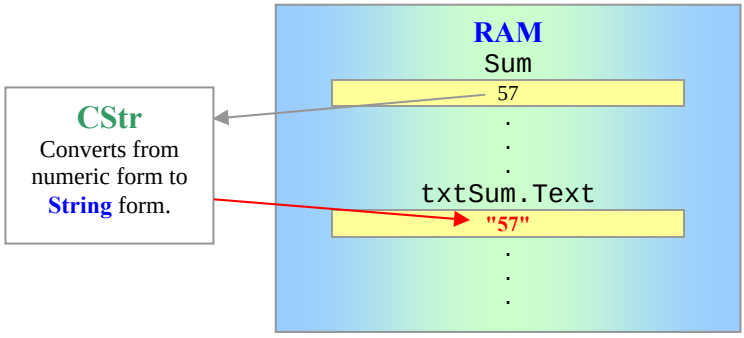
3. The next statement actually calculates the sum of the two numbers entered by the user.

```
'PROCESSING: Calculate the sum and assign to "Sum"
Sum = Number1 + Number2
```



4. Finally, the output is displayed by setting the value of the "Text" property of "txtSum" equal to the value of "Sum." (The value of "Sum" must first be converted to **String** (text) form before it can be assigned to the "Text" property of "txtSum.")

```
'OUTPUT: Display results.
txtSum.Text = CStr(Sum)
```



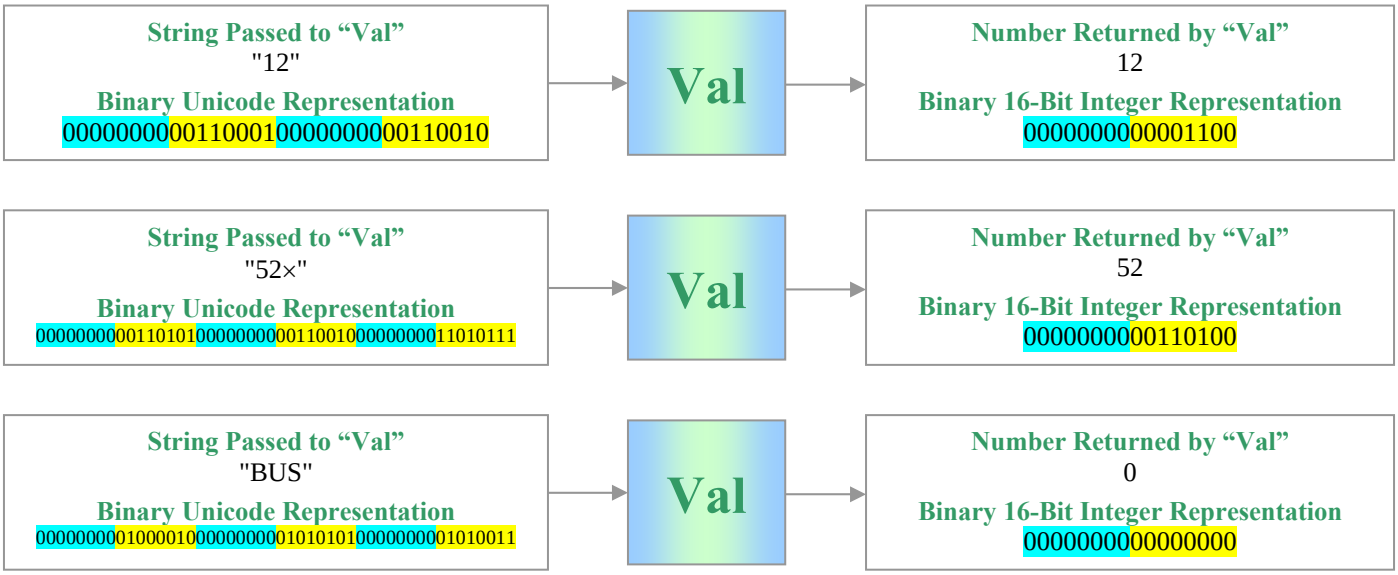
Questions

1. Load the addition calculator program from **I:\Out\Nolfi\Ics3mo\Simple VB Examples\Addition Calculator**. Edit the code by deleting the "Val" function. For example, use the statement "`Number1 = txtNumber1.Text`" instead of "`Number1 = Val(txtNumber1.Text)`." Then run the program and experiment by entering both numeric and non-numeric values. What happens when you enter non-numeric values? Does this problem still occur if you use the "Val" function?
2. Modify the addition calculator program in such a way that it is also able to perform subtraction, multiplication and division. **Note**: It is important that you use terminology correctly. "Sum" refers to the quantity obtained by ***adding a group of numbers***. You should use the terms ***difference***, ***product*** and ***quotient*** for subtraction, multiplication and division respectively.

A Closer Look at "Val" and "CStr"

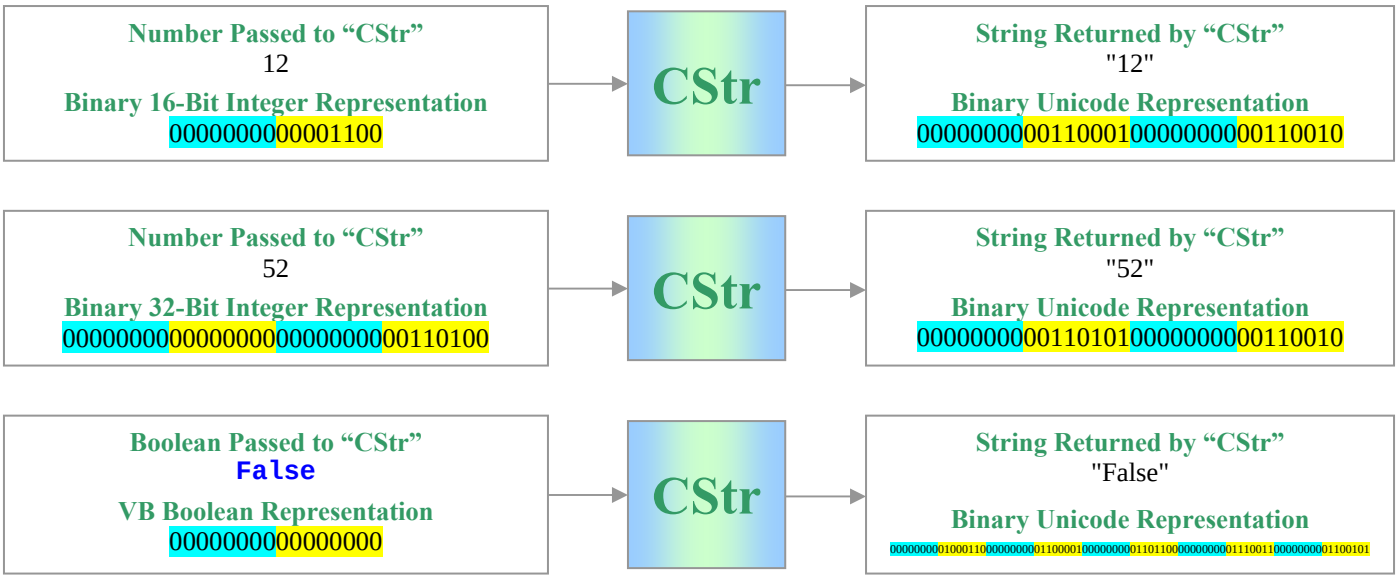
The "Val" Function

As we have learned, the "Val" function is used to convert a ***string value*** to a ***numeric value***. As the examples below show, the "Val" function scans the given string character-by-character from left to right. As soon as a ***non-digit*** is found or the ***end of the string is reached***, Val halts its search and returns its result. The result is the numeric value of the string, represented ***using an appropriate numeric encoding scheme***.



The "CStr" Function

The "CStr" function is used to convert ***any value*** to a ***string value***. The "CStr" function always returns a string consisting of Unicode characters. Some examples are shown below.



A Program that Processes String (Text) Information

Introduction

The main purpose of the previous programming example was to show how a computer can process ***numeric information*** using mathematical operations. The following example shows how computers can process ***text*** (e.g. words, addresses, phone numbers, etc).

In the first unit of this course, we encountered the idea of a ***numeric constant***. If, for example, we needed to draw several circles with a constant radius of 10 units, we could use a statement such as

```
Me.Circle (X, X), 10
```

Numeric Variable ***Numeric Constant***

The example given below will introduce the ideas of ***string variables*** and ***string constants***. String variables work in much the same way as any other variables. The only difference is that they are declared as type "**`String`**" instead of some numeric type. String constants, on the other hand, look very different from numeric constants. As you will see in the following example, string constants are always enclosed in quotation marks.

The String (Text) Processing Example

You will find the following program in the folder

I:\Out\Nolfi\Ics3mo\Example Programs\Simple VB Examples\Friendly Message

Load the program, experiment with it and study its code. Then answer the questions on the next page.

```
Option Explicit

Private Sub cmdMessage_Click()

    'MEMORY: Variable declaration
    Dim UserName As String

    'INPUT
    UserName = Trim(txtName.Text)

    'OUTPUT
    lblMessage.Caption = "Hi " & UserName & "! I'm glad you can spell your name. " _
                & "Your name can't possibly be Balraj since you know how to spell! " _
                & "It was nice meeting you " & UserName & ". Bye!"

End Sub
```

This operator is called the ***string concatenation operator***. Its purpose is to create a new string by ***joining*** one string to another. Although the symbol "&" is called the "ampersand" and it is often used as an abbreviation of the word "and," its meaning in VB is entirely unrelated to the word "and."

This combination of a space followed by an underscore is used in VB to spread out very long statements over two or more lines of code.

String Variable

String Constant

Extremely Important Questions

1. The first statement in every VB program should be "`Option Explicit`." What is its purpose? How does it help you to ***debug*** your programs? What can go wrong if you forget to include it?

2. An apostrophe (single quotation mark) is used to begin certain statements in VB. (The word "Rem" can also be used to begin this type of statement.) What are such statements called? What is their purpose? How does the computer process such statements? How can these statements be used to remove a statement from a program without deleting it?

3. A "`Sub`" is a program ***subroutine***, that is, a ***portion of a program that is named*** so that it can be accessed whenever needed. The "`Sub`" shown above is automatically named "cmdMessage_Click" when you double click the "cmdMessage" command button. Explain how VB determines this name.

4. The statement "`Dim UserName As String`" is used to ***declare the variable*** "UserName." The ***name*** of the variable being declared is _____ . Its ***type*** is _____, which means that it is used to store _____ information. Declaring variables helps programmers to _____ their programs, allows an operating system to determine how much _____ is needed to store the values of the variables and which _____ scheme to use, and it helps to determine which _____ can be used to process information of a given _____.

5. The statement "`UserName = Trim(txtName.Text)`" is called an ***assignment statement*** because it is used to ***assign*** (give) a ***value*** to a ***variable***. Complete the following:

   Name of the variable being assigned a value: _____

   Name of the object from which a property is being used in the assignment statement _____

   Name of the property whose value is being assigned to the variable: _____

   Purpose of the "Trim" intrinsic (built-in) function: _____

   _____

6. Explain the difference between the ***name of a variable*** and the ***value of a variable***. Give an example to illustrate your answer.

7. Explain the difference between the ***name of an object*** and the ***name of a variable***. Give an example to illustrate your answer.

8. What is the purpose of the "&" operator? What is it called? To what ***type of data*** does it apply? Why is it inappropriate to call it an "and" or an "ampersand" in the context of VB?

9. What is the purpose of ***quotation marks*** in VB programs? What will happen if you forget to use quotation marks when they are needed? What will happen if you use quotation marks when they are ***not*** needed?

10. What is the purpose of using a space followed by an underscore? Why is this useful?

How Computers make Decisions (Selections)

Introduction to "If" Statements

So far we have only considered programs in which the next statement to be executed immediately follows the previously executed statement. However, there are many circumstances under which the next statement to be executed will depend on a user action, a system event or some other unpredictable occurrence. In such cases, programs must ***select*** a statement or a group of statements and ***reject*** others. In VB this is accomplished through "**If**" statements. Study the following program carefully. It can be found in the folder

I:\Out\Nolfi\Ics3m0\Simple VB Examples\Friendly Message - Sneaky Version

```
Private Sub cmdMessage_Click()

    'MEMORY: Variable declaration
    Dim UserName As String, LowerCaseUserName As String, Message As String

    'INPUT
    UserName = Trim(txtName.Text)

    'PROCESSING
    LowerCaseUserName = LCase(UserName) 'Store lower case copy of the user's name.

    'Based on the name entered by the user, create an appropriate message.
    If LowerCaseUserName = "balraj" Then
        Message = "Hi Balraj! I'm glad that you have learned to spell your name. Unfortunately," _
                & " some people still think that your name is  spelled 'Balrash.'"
    ElseIf LowerCaseUserName = "nolfi" Then
        Message = "Greetings, oh great master and creator of lowly programs like me!" _
                & " As usual, it is an honour to be in your presence!"
    ElseIf LowerCaseUserName = "maham" Then
        Message = "'HEY! Nolfi wasn't supposed to hear that comment! I suppose that he" _
                & " can't help it. After all, I am very LOUD whenever I make such remarks!" _
                & " Pssst, Khyandra. Isn't he supposed to be deaf by his age?'"
    ElseIf LowerCaseUserName = "abhay" Or LowerCaseUserName = "chad" Then
        Message = "HUH?????"
    ElseIf LowerCaseUserName = "matt" Then
        Message = "I defy all those who attempt to coerce me to do school work." _
                & "NEVER! NEVER! Long live the 'World of Warcraft.'"
    Else
        Message = "Hi " & UserName & "! I'm glad you can spell your name. " _
                & "Your name can't possibly be Balraj since you know how to spell! " _
                & "It was nice meeting you " & UserName & ". Bye!"
    End If

    'OUTPUT
    lblMessage.Caption = Message
End Sub
```

"LCase" is an intrinsic function that converts a string to lower case. "UCase" converts a string to upper case.

Indentation Margin Lines

If Statement Details

If statements are used in programs to make **decisions** or **selections.** The rules for **If** statements are as follows:

- **If** statements begin with the word **If** and end with the words **End If**
- There must be **exactly one If** and **one End If**
- There may be **zero** or **more ElseIf** clauses. **ElseIf** clauses must follow **If** and precede **Else**.
- Both **If** and **ElseIf** clauses must have a **condition** and must have the keyword **Then.**
- There may be **zero Else** clauses or **one Else** clause. **Else** must follow **If** and **ElseIf**, and **Else must not** have a **condition** or the keyword **Then**. **Else** means "if all else fails."

- There must be **exactly one If.**
- There may be **zero** or **more ElseIf** clauses.
- There may be **zero** or **one Else** clauses.
- There must be **exactly one End If** (to mark the end of the selection structure).

General Structure of an If Statement

If *condition* **Then**
statements
ElseIf *condition* **Then**
statements
.
.
.
ElseIf *condition* **Then**
statements
Else
statements
End If

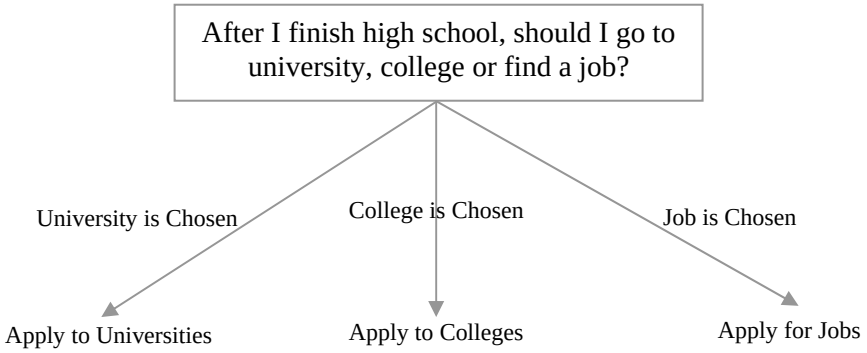
- A ***condition*** is formed by using ***conditional operators*** such as =, <, >, <=, >=, < >, **Is** and **Like**.

  e.g. If Age >= 19 **Then**

- ***Conditions*** can be combined by using ***logical operators*** such as **And, Or** and **Not.**
- The term ***statements*** refers to any group of valid VB statements. Notice that the statements are further ***indented*** one TAB ***within*** **If** statements. The statements that begin with **If**, **ElseIf** and **Else** are **NOT** indented further.

Picturing "If" Statements

The following diagram can be useful in understanding the flow of information during the execution of an "**If**" statement is executed. "If" statements are a lot like travelling along a path and suddenly reaching a "fork." When this happens, a ***decision*** needs to be made.



Exercises

1. Write a program that allows a user to enter a mark in an input box. The program then displays "Congratulations you have PASSED," or "Sorry, you have FAILED" in a ***label*** depending on whether the mark is greater than or equal to 50 or less than 50.

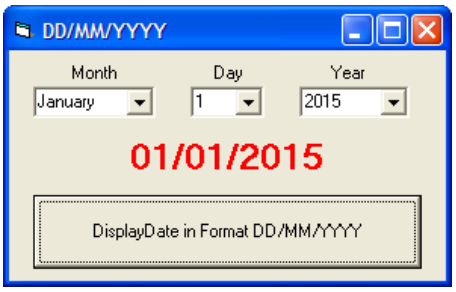
2. Most universities in North America use a grading system known as the GPA (grade point average) system. It is summarized in the table given below.

Percentage Grade	*Grade Point Score*
85% – 100%	4.0
80% – 84%	3.7
77% – 79%	3.3
74% – 76%	3.0
70% – 73%	2.7
67% – 69%	2.3
64% – 66%	2.0
60% – 63%	1.7
57% – 59%	1.3
54% – 56%	1.0
50% – 53%	0.7
0% – 49%	0.0

For a solution to this problem, see
I:\Out\Nolfi\Ics3m0\GPA Solution

Write a VB program that displays the grade point score given the percentage grade. In addition, your program should display an error message for invalid percentage grades (i.e. grades lower than 0% or higher than 100%).

3. Copy the contents of the folder **I:\Out\Nolfi\Ics3m0\If Statement Example – Date** to your "g:" drive. Within this folder you will find a VB project file called "Date.vbp." Load the "Date.vbp" project and experiment with it for a few minutes. You will discover that three ***combo boxes*** are used to allow the user to select the month, day and year. (A combo box combines the functionality of a text box with that of a list box.)



When you examine the VB code for this project, it may look very complicated to you. Please do not be discouraged by the appearance of the code! All you need to do is write the code for the command button. That is, you must write code that takes the date given by the values stored in the combo boxes and converts it to the format DD/MM/YY (2 digits for the day, 2 digits for the month and four digits for the year).

Note: Although it is not required at this point, students who are confident enough may wish to study the code given in this project. Since this program contains a plethora of new ideas to explore, it is possible to learn a great deal from it!

4. **Here is the game of GREED v1.0.** The player clicks **New Game** and then the dice are allowed to roll. The idea of the game is to make as much money as possible.

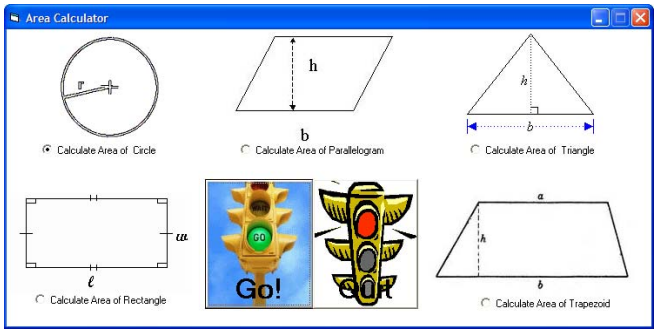
Your **FIRST** roll is recorded and if at any time during the game you roll that number again, **you lose everything**. Each time you click **ROLL**, and the roll is ***not*** the same as the **FIRST** roll, you **double your money**. You can click **STOP** at any time and you **keep** the money you have earned.

Use the following code to generate two random integers between 1 and 6 and store the results using the variable names "Die1" and "Die2." How this code works will be explained later in the unit.

```
Die1 = Int(Rnd*6+1)
Die2 = Int(Rnd*6+1)
```

You can find a very sophisticated solution to this problem in the folder
I:\OUT\Nolfi\Ics3m0\Game of Greed - Enhanced Version

Another Program that Requires "If" Statements



The "Area Calculator" program can be found in **I:\Out\Nolfi\Ics3m0\Area Calculator**. Load this program and study the code carefully. Notice that an "**If**" statement is used to determine the shape that has been selected by the user.

```
Private Sub cmdGo_Click()
  If optRectangle.Value = True Then
    frmChosenShape.imgShape.Picture=imgRectangle.Picture
    frmChosenShape.Caption = "Area of Rectangle"
    frmChosenShape.lblDimension1.Visible = True
    frmChosenShape.lblDimension2.Visible = True
    frmChosenShape.lblDimension3.Visible = False
    frmChosenShape.lblDimension1.Caption = "l="
    frmChosenShape.lblDimension2.Caption = "w="
    frmChosenShape.lblDimension3.Caption = ""
    frmChosenShape.txtDimension1.Visible = True
    frmChosenShape.txtDimension2.Visible = True
    frmChosenShape.txtDimension3.Visible = False
    frmChosenShape.Show
  ElseIf optParallelogram.Value = True Then
    frmChosenShape.imgShape.Picture = imgParallelogram.Picture
    frmChosenShape.Caption = "Area of Parallelogram"
    frmChosenShape.lblDimension1.Visible = True
    frmChosenShape.lblDimension2.Visible = True
    frmChosenShape.lblDimension3.Visible = False
    frmChosenShape.lblDimension1.Caption = "b="
    frmChosenShape.lblDimension2.Caption = "h="
    frmChosenShape.lblDimension3.Caption = ""
    frmChosenShape.txtDimension1.Visible = True
    frmChosenShape.txtDimension2.Visible = True
    frmChosenShape.txtDimension3.Visible = False
    frmChosenShape.Show
  ElseIf optTriangle.Value = True Then
    frmChosenShape.imgShape.Picture=imgTriangle.Picture
    frmChosenShape.Caption = "Area of Triangle"
    frmChosenShape.lblDimension1.Visible = True
    frmChosenShape.lblDimension2.Visible = True
    frmChosenShape.lblDimension3.Visible = False
    frmChosenShape.lblDimension1.Caption = "b="
    frmChosenShape.lblDimension2.Caption = "h="
    frmChosenShape.lblDimension3.Caption = ""
    frmChosenShape.txtDimension1.Visible = True
    frmChosenShape.txtDimension2.Visible = True
    frmChosenShape.txtDimension3.Visible = False
    frmChosenShape.Show
  ElseIf optCircle.Value = True Then
    frmChosenShape.imgShape.Picture=imgCircle.Picture
    frmChosenShape.Caption = "Area of Circle"
    frmChosenShape.lblDimension1.Visible = False
    frmChosenShape.lblDimension2.Visible = True
    frmChosenShape.lblDimension3.Visible = False
    frmChosenShape.lblDimension1.Caption = ""
    frmChosenShape.lblDimension2.Caption = "r="
    frmChosenShape.lblDimension3.Caption = ""
    frmChosenShape.txtDimension1.Visible = False
    frmChosenShape.txtDimension2.Visible = True
    frmChosenShape.txtDimension3.Visible = False
    frmChosenShape.Show
  ElseIf optTrapezoid.Value = True Then
    frmChosenShape.imgShape.Picture=imgTrapezoid.Picture
    frmChosenShape.Caption = "Area of Trapezoid"
    frmChosenShape.lblDimension1.Visible = True
    frmChosenShape.lblDimension2.Visible = True
    frmChosenShape.lblDimension3.Visible = True
    frmChosenShape.lblDimension1.Caption = "a="
    frmChosenShape.lblDimension2.Caption = "b="
    frmChosenShape.lblDimension3.Caption = "h="
    frmChosenShape.txtDimension1.Visible = True
    frmChosenShape.txtDimension2.Visible = True
    frmChosenShape.txtDimension3.Visible = True
    frmChosenShape.Show
  Else
    MsgBox "Please select one of the shapes before clicking 'Go!'",  vbExclamation
  End If
End Sub
```

Questions

1. The area calculator program uses two forms, one that is used to select the shape and another that is used to allow the user to enter the dimensions of the shape. How is this accomplished?
2. Once the user chooses a shape and clicks "Go," another form is displayed to allow the user to enter the dimensions of the shape. How would you prevent the user from returning to the original form (the ***parent form***) unless the new form (the ***child form***) is first closed?

Overview: Sequence, Selection and Repetition: The Underpinnings of Programming

Sequence

Instructions are executed (carried out) in **sequence** (in order, one after the other). All statements are executed exactly once; none of the statements is omitted.

Example

```
' Friendly greeting program
Option Explicit
Private Sub cmdPressMe_Click()
   'Memory
   Dim FirstName As String
   'Input
   FirstName = Trim(txtName.Text)
   'Processing and Output
   lblGreeting.Visible = True
   lblGreeting.Caption = "Have a " & _
         "nice day " & FirstName & "!"
End Sub
```

Selection

Based on a condition or a set of conditions, certain statements are ***selected*** while others are rejected. The idea of ***selection*** should be used whenever your program needs to make a **decision.**

Example

```
' Which number is larger?
Option Explicit
Private Sub cmdLarger_Click()
    'Memory
    Dim Num1 As Double, Num2 _
                As Double, _
                Larger As Double
    'Input
    Num1 = Val(txtNum1.Text)
    Num2 = Val(txtNum2.Text)
    'Processing
    If Num1 > Num2 Then
        Larger = Num1
    Else
        Larger = Num2
    End If
    'Output
    lblLarger.Caption = CStr(Larger)
End Sub
```

Repetition

Whenever your program needs to **repeat** certain instructions two or more times, the concept of ***repetition*** (looping) is used. Many different types of **loops** can be constructed, depending on the particular situation.

Example

```
' Program to add the cubes of
' the numbers from 1 to 5
Option Explicit
Private Sub cmdSumOfCubes_Click()
    'Memory
    Dim I As Byte
    Dim Total As Double
    'Processing
    Total = 0
    'I is called a loop counter
    'variable
    For I = 1 To 5
        Total = Total + I ^ 3
    Next I
    'Output
    lblSum.Caption = CStr(Sum)
End Sub
```

Very long VB statements are easier to read if they are broken up into two or more physical lines. To do this, use the **statement continuation character** " _" (a space followed by an underscore).

Notice the **indentation** used in these programs. Although your programs will work without proper indentation, they will be extremely difficult to read, understand and debug. The rules of indentation are simple and **must be observed by all students.** Failing to indent properly will result in a significant loss of marks. **RULES OF INDENTATION:** Indent one tab space within ***subs***, ***if statements*** and ***loops*** (more details will be given in subsequent examples).

Questions and Programming Exercises

1. What is the purpose of the statement continuation character?
2. Why is it important to indent programs properly?
3. Explain the terms ***sequence***, ***selection*** and ***repetition***.
4. Define the term ***underpinning***.
5. To understand the example of repetition given above, it is very helpful to trace the execution of the program by using something called a **memory map**. A memory map is simply a table that displays the changing values of variables. Complete the memory map shown below.

Before Loop

Each of these Rows Shows Values of Variables ***after*** each Repetition

After Loop

I	I^3	Total
0	0	0
1	1	1
2	8	9
3		
4		
5		
6		

6. In the example shown above for repetition, you will find the assignment statement **Total = Total + I^3.** Since you are accustomed to mathematical equations, you may misinterpret this Visual Basic statement. In Visual Basic, the statement above should be interpreted as follows:

Total = Total + I ^ 3

(The new value of the variable 'Total') **is assigned the value of** (The current value of the variable 'Total') **plus** (I to the exponent 3)

Now consider the mathematical equation $x = x + 3$. How does the meaning of this equation differ from that of the assignment statement shown above? Does this mathematical equation have a solution? Explain.

7. Consider the "sum of the cubes" program given on the previous page (in the "Repetition" column of the table).

   1. so that it can calculate the sum of the cubes from **Lowest** to **Highest**, where Lowest and Highest are integer values. To prevent numeric overflow errors, think carefully about the type of the **Total** variable.

8. Write Visual Basic programs that use "**For**" loops to

(a) print the following on your form

```
*****     *****     *********
*****     *****     *********
*****     *****     *********
*****     *****     *********
*****     *****     *********
****************    *********
****************    *********
****************    *********
*****     *****     *********
*****     *****     *********
*****     *****     *********
*****     *****     *********
*****     *****     *********
```

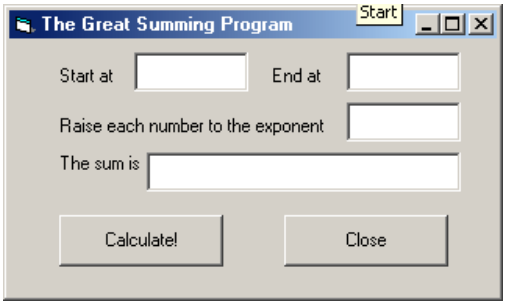
(b) fill your form with asterisks (i.e. *)

(c) find the sum of the numbers from 1 to 1000

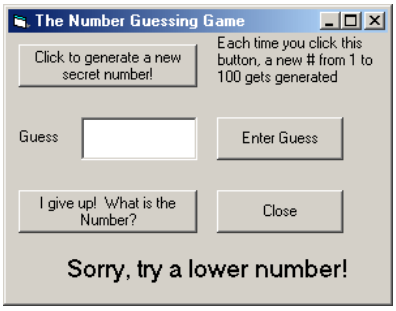
(d) find the sum of the ***even*** numbers from 2 to 1000

(e) find the sum of the squares of the numbers from 1 to 1000 (**Note:** The **Integer** data type does not have a large enough range for this program. Try **Long** instead.)

9. Modify further the program in question 8 so that it can calculate the sums of consecutive numbers to any exponent. Do not expect your program to work for all values that you enter. Remember that like your calculators, computers can only represent numbers that are so large or so small. Try different values to find out the limitations of your program.



10. Write a Visual Basic program for a number guessing game. Your program should generate a random integer between 1 and 100. Then the user keeps guessing until the number is found or until the "I give up" button is clicked. Each time the user enters an incorrect guess, your program should indicate whether the secret number is higher or lower. If the guess is correct, your program should output a congratulatory message.



NOTE: Use the VB code **SecretNumber = Int (Rnd * 100 + 1)** to generate the secret numbers. If you are observant, you will notice that your game will be very predictable. We shall soon discuss a solution to this problem.

Using VB to Generate Pseudo-Random Numbers

Introduction

Without an element of randomness, many computer applications would be extremely dull. Can you imagine playing your favourite video game if there were no surprises whatsoever? The evil enemy would always appear at exactly the same place and time and the outcome of every battle would be tiresomely predictable. Under such conditions, would it still be your favourite game? Luckily, ***pseudo-random numbers*** come to the rescue! The ***unpredictability*** of our favourite games is due entirely to a computer's ability to generate sequences of ***seemingly*** random numbers.

Why Pseudo?

It is ***not possible*** for a computer to generate random numbers, at least not in the strictest sense of the word "random." Since computers can only function by following the steps in algorithms, it follows that computers can only produce numbers that result from the execution of algorithms. Clearly, there is nothing random about this process because the steps of any algorithm can be carried out by anyone who knows the algorithm. Therefore, it appears that we are trapped in a vicious circle. Computers cannot function without algorithms but the output of any algorithm is, at least in theory, completely predictable. How then, can randomness spring from predictability?

Fortunately, there is a way to resolve this conundrum. Computers can ***simulate randomness*** by executing algorithms that produce sequences of numbers that ***cannot be distinguished from*** true sequences of random numbers. Such algorithms are known as ***pseudo-random number generators***.

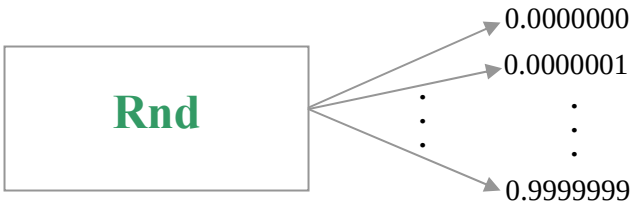
pseudo-: false, counterfeit, fake, sham, deceptive

Other words beginning with the prefix "pseudo-"

pseudonym, pseudoscience, pseudopod, pseudocode

How to Generate Pseudo-Random Numbers in VB

"Rnd" is an intrinsic function in VB that generates pseudo-random numbers greater than or equal to zero and less than one. In other words, "Rnd" produces a pseudo-random **Single** value as low as 0.0000000 and as high as 0.9999999.



By applying appropriate transformations, we can use "Rnd" to generate pseudo-random NUMBERS in any range.

Complete the following. The first one is done for you.

1. **Rnd*2** generates a pseudo-random number that is greater than or equal to __0__ and less than to __2__ .

2. **Rnd*100** generates a pseudo-random number that is greater than or equal to _____ and less than _____.

3. **Rnd*9+7** generates a pseudo-random number that is greater than or equal to _____ and less than _____.

4. **Rnd*6–5** generates a pseudo-random number that is greater than or equal to _____ and less than _____.

5. **Rnd*3+1.5** generates a pseudo-random number that is greater than or equal to _____ and less than _____.

6. **Rnd*6–0.5** generates a pseudo-random number that is greater than or equal to _____ and less than _____.

By applying the "Int" intrinsic function along with "Rnd" and appropriate transformations, we can generate pseudo-random INTEGERS in any range. The "Int" intrinsic function ROUNDS DOWN to the nearest integer.

Examples – Expressions Involving "Int"

Int(3.9) = 3 Int(3.1) = 3 Int(4) = 4 Int(–3.9) = –4 Int(–3.01) = –4

Complete the following. The first two are done for you.

1. **Int(Rnd*2)** generates a pseudo-random integer in the range 0, 1 .
2. **Int(Rnd*100)** generates a pseudo-random integer in the range 0, 1, 2, 3, … , 97, 98, 99 .
3. **Int(Rnd*100+1)** generates a pseudo-random integer in the range _____.
4. **Int(Rnd*6–5)** generates a pseudo-random integer in the range _____.
5. **Int(Rnd*6)** generates a pseudo-random integer in the range _____.
6. **Int(Rnd*6+1)** generates a pseudo-random integer in the range _____.
7. **Int(Rnd*100–50)** generates a pseudo-random integer in the range _____.
8. **Int(Rnd*1000+1)** generates a pseudo-random integer in the range _____.
9. **Int(Rnd*1001+1)** generates a pseudo-random integer in the range _____.
10. **Int(Rnd*** [ ] + [ ] **)** generates a pseudo-random integer in the range 1, 2, 3, 4, … , 9998, 9999, 10000.

A General Expression for Generating Pseudo-Random Integers in VB

Based on your answers to questions 1 to 10 above, complete the following.

To generate a random integer greater than or equal to "`Lowest`" and less than or equal to "`Highest`," use the expression

`Int(Rnd *` [ ] **+** [ ] **`)`**

Questions

Write VB expressions to generate pseudo-random ***integers*** in each of the following ranges.

1. From 1 to 6: _____
2. From 0 to 5: _____
3. From –5 to 5: _____
4. From 1 to 999: _____
5. From 1 to 1000: _____
6. From –5000 to 10000: _____

Applying Pseudo-Random Integers – An Enhanced Version of the Game of Greed

Instructions

Load the enhanced version of the "Game of Greed" found in **I:\Out\Nolfi\Ics3m0\Game of Greed - Enhanced Version**. Study the program carefully and then answer the following questions. Note that this program contains a few advanced programming concepts that we have not yet learned. Do not be deterred by this. The main point of this exercise is to understand how pseudo-random numbers can make programs more versatile and more interesting.

Questions

1. Explain the difference between random numbers and pseudo-random numbers.

2. The following statements are used to generate the pseudo-random integers for the dice roll:

```
'Generate two random integers beween 1 and 6 inclusive
Die1 = Int(Rnd * 6 + 1)
Die2 = Int(Rnd * 6 + 1)
```

Why would it be incorrect to replace these two statements with the following single statement?

```
Roll = Int(Rnd*11 + 2) 'This statement generates a pseudo-random integer from 2 to 12 inclusive
```

3. The purpose of this question is to understand the importance of using the "Randomize" statement in VB programs that use the "Rnd" intrinsic function.

(a) You will notice that the "Game of Greed" code includes a sub called "Form_Load." Explain how such subs behave and when it is appropriate to use them.

(b) You will also notice that the "Form_Load" sub contains the "Randomize" statement. To understand the importance of this statement, remove it temporarily by turning it into a comment as shown below.

```
'Randomize
```

Now play the game several times and take careful note of the rolls that are generated. What do you notice? Use your observations to explain the purpose of the "Randomize" statement.

(c) Now use "MSDN" help to look up the "Randomize" statement. Explain the meaning of the term "seed."

4. You will notice that certain local variables in the "Game of Greed" are declared using the keyword "**Static**" instead of the keyword "**Dim**." Explain the difference between the two types of declarations.

5. What is the purpose of the "DoEvents" statement? What happens if you delete the "DoEvents" statement from the loop found within the "cmdRoll_Click" sub?

6. "DoEvents" should be used with caution because it can cause problems. In the "cmdRoll_Click" sub, temporarily remove the statement "`cmdRoll.Enabled = False`" by turning it into a comment (shown below).

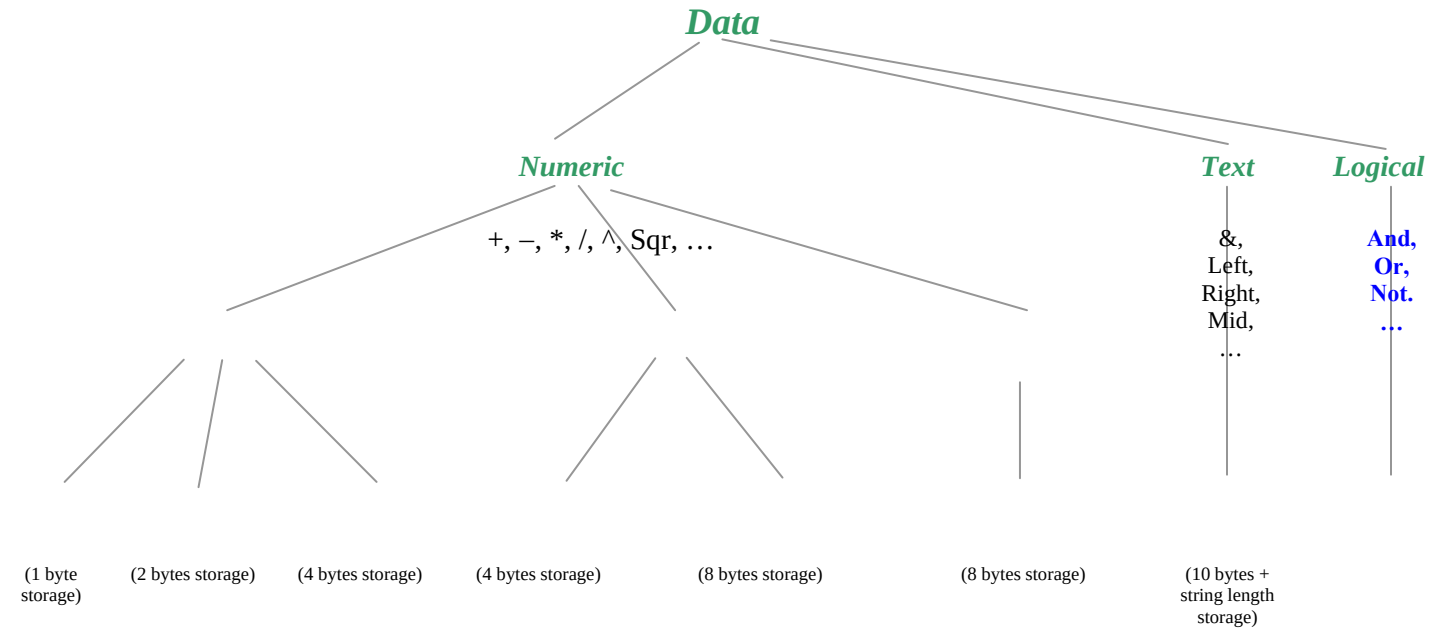
```
'cmdRoll.Enabled = False
```

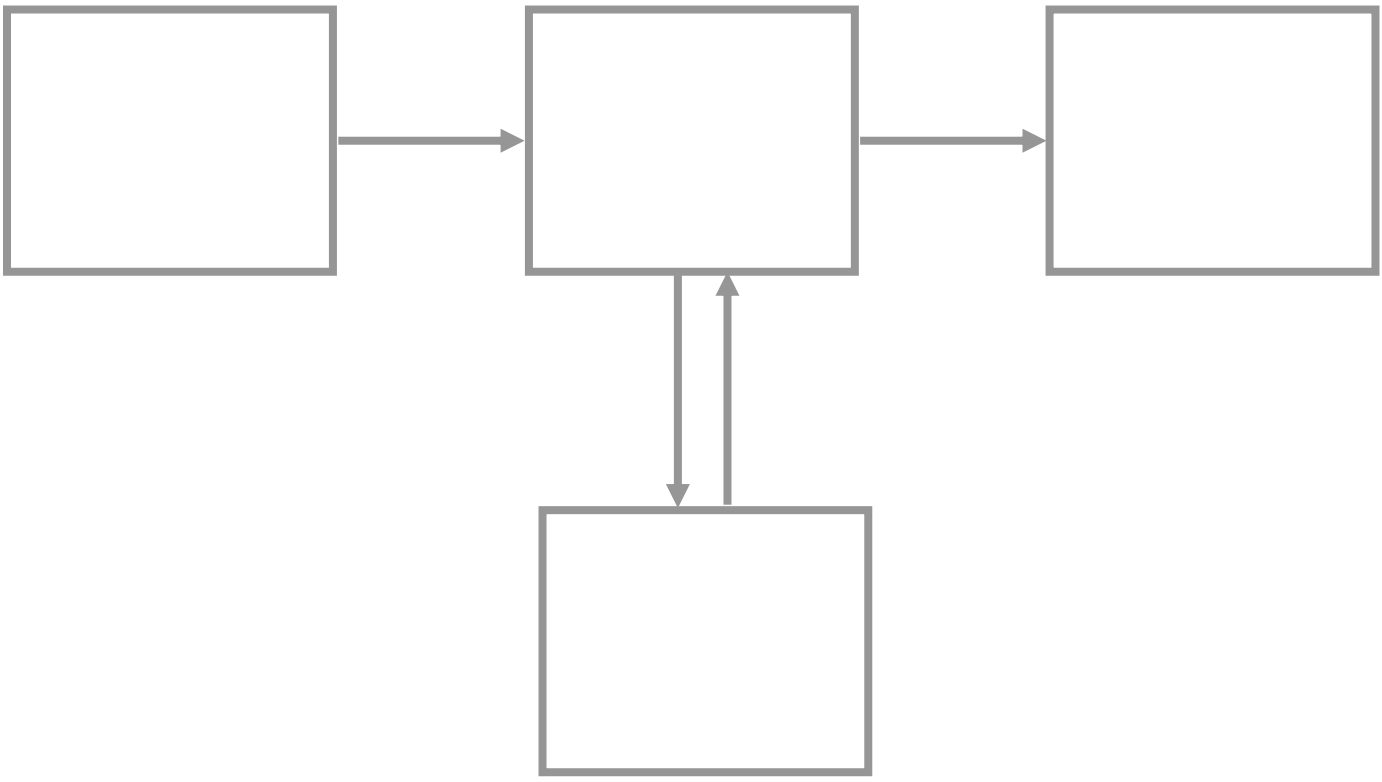
Then play the game and click the "Roll" button repeatedly while the dice animation is running. What happens? Explain what causes this strange behaviour.

ICS3M0 – Review of First Half of Unit 2

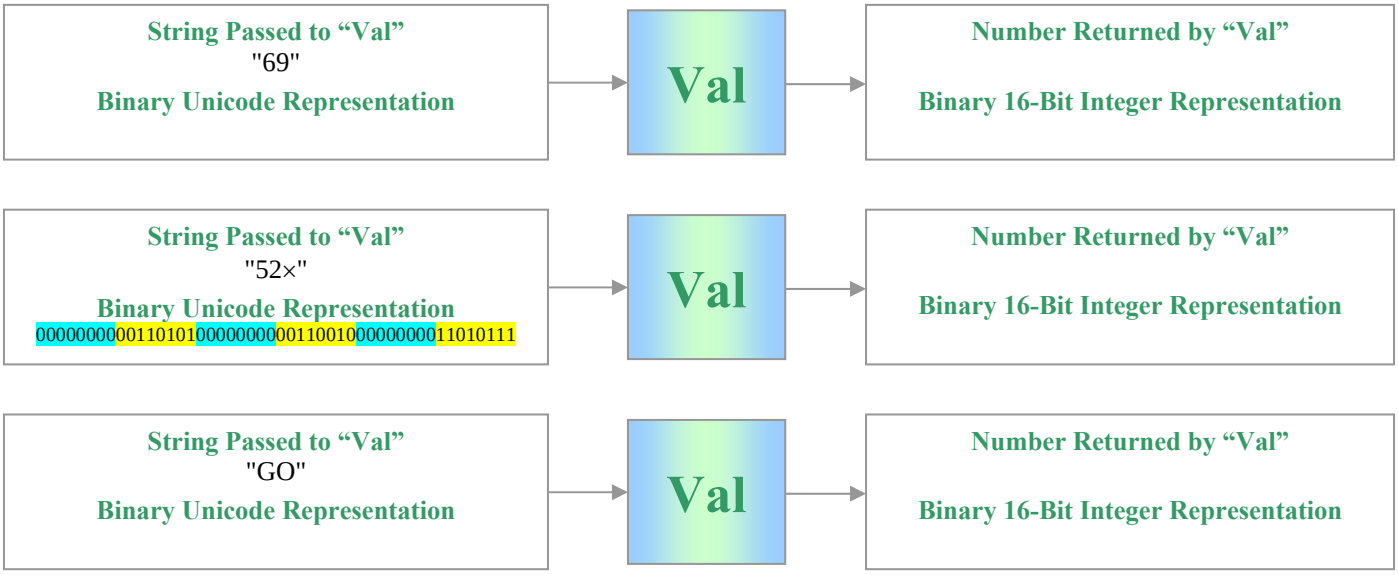
Data Types

1. Complete the following diagrams.

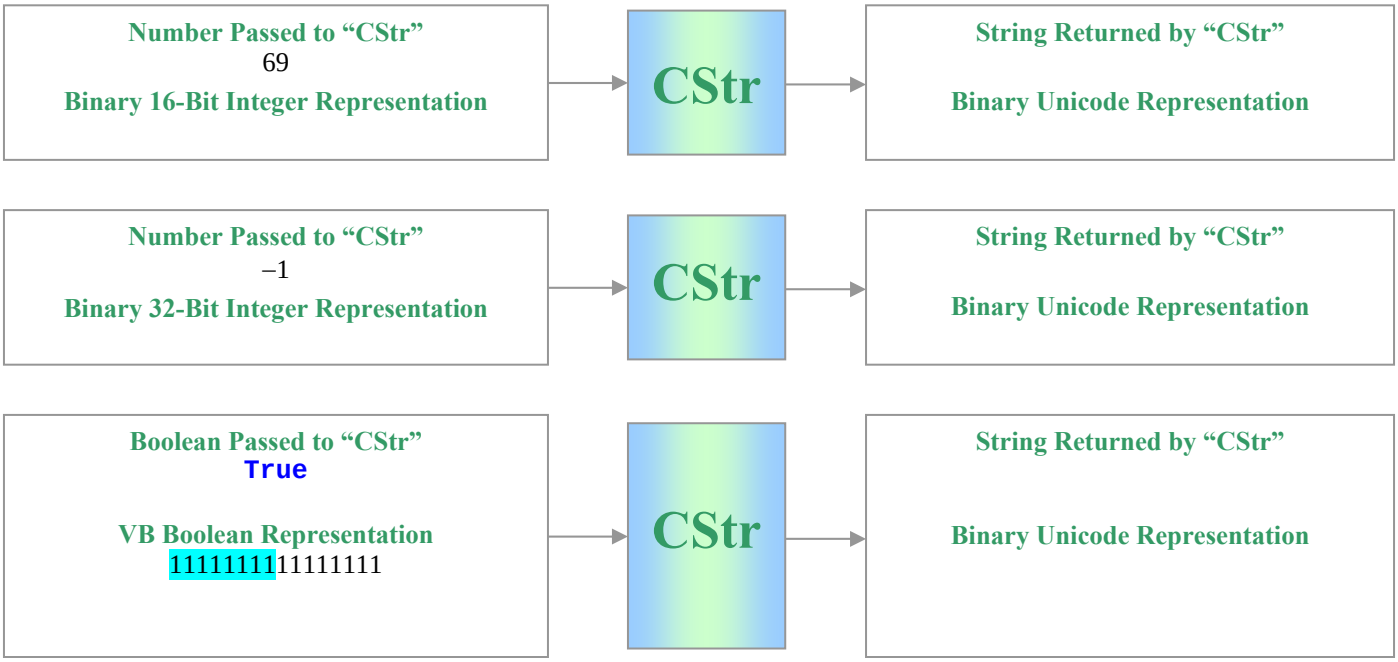




The “Val” function is used to convert a _____ to a _____. As the examples below show, the “Val” function scans the given string _____ from left to right. As soon as a _____ is found or the _____, Val halts its search and returns its result. The result is the numeric value of the string, represented _____.



2. The “CStr” function is used to convert _____ to a _____. The “CStr” function always returns a string consisting of _____ characters.



Using VB to Generate Pseudo-Random Numbers

1. Why are random numbers produced by a computer called "pseudo-random numbers?"

2. Why is it important in software development to be able to generate random integers?

3. **Int(Rnd*50+20)** generates a pseudo-random integer in the range _____.

4. **Int(Rnd*10–8)** generates a pseudo-random integer in the range _____.

5. **Int(Rnd*20)** generates a pseudo-random integer in the range _____.

6. **Int(Rnd*6+1)** generates a pseudo-random integer in the range _____.

7. **Int(Rnd*100–50)** generates a pseudo-random integer in the range _____.

8. **Int(Rnd*** ☐ **+** ☐ **)** generates a pseudo-random integer in the range –10, –9, –8, … , 13, 14, 15.

9. **Int(Rnd*** ☐ **+** ☐ **)** generates a pseudo-random integer in the range Lowest, …, Highest.

10. Write VB expressions to generate pseudo-random ***integers*** in each of the following ranges.

(a) From 1 to 6: _____

(b) From 0 to 50: _____

(c) From –15 to 25: _____

(d) From 1 to 9999: _____

(e) From 1 to 10000: _____

(f) From –5000 to 10000: _____

"If" Statements

Write a VB program that displays a friendly message based on the temperature, in degrees Celsius, entered by the user. Here is a list of suggested temperature ranges and one suggested message. Feel free to modify them as you see fit.

Temperature Entered	*Message*
Below –40º C	Get inside before some important body parts freeze off!
–40º C to –20º C	
–20º C to 0º C	
0º C to 10º C	
10º C to 20º C	
20º C to 30º C	
30º C to 40º C	
Above 40º C	

Problem Solving Strategy 1:
Solve a Complex Problem by Investigating Specific Examples of the Problem

Case Study 1: Time Converter Problem

General Problem Statement

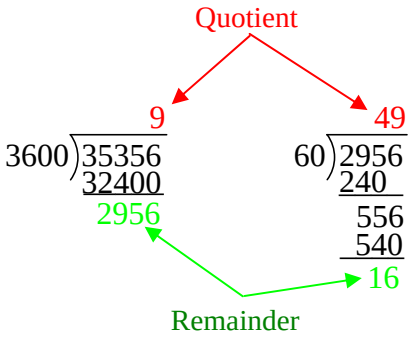
Input a value specified in seconds and convert to hours, minutes and seconds.

Where Should I Begin?

If you do not know how to select a strategy for solving this (or any other) problem, examining a specific example often helps to shed some light on the situation.

e.g. Convert 35356 s to the format *h : min : s*.

	hours	minutes	seconds
Step 1	0	0	35356
Step 2	9	0	2956
Step 3	9	49	16



35356 ÷ 3600 = **9** R **2956**

2956 ÷ 60 = **49** R **16**

Questions

1. Why was 35356 divided by 3600? Why was 2956 divided by 60?

2. Explain how Visual Basic can be used to compute a quotient and a remainder. (Use Google to find an answer to this question if you don't know the VB operators used to find quotient and remainder. Also, near the bottom of this page, you will find an "upside-down answer" to this question.)

Writing an Algorithm

1. The user enters a time in seconds: ***seconds***
2. Set ***hours*** to the quotient of ***seconds*** divided by 3600
3. Set ***seconds*** to the remainder of ***seconds*** divided by 3600
4. Set ***minutes*** to the quotient of ***seconds*** divided by 60
5. Set ***seconds*** to the remainder of ***seconds*** divided by 60
6. The result is ***hours*** : ***minutes*** : ***seconds***

Writing the Visual Basic Code

- Use the **Mod** operator to evaluate the **remainder**
- Use the "\" operator (integer division) to evaluate the **quotient**
- Do not confuse "\" (integer division) with "/" (floating point division). The designers of VB made a poor decision choosing the backslash as the symbol for integer division. It is too easily confused with the forward slash.

Exercises

1. Convert 234567 seconds to the format *h : min : s*.

2. Convert 8999.78 minutes to the format *h : min : s*.

3. Convert 84.69 hours to the format *h : min : s*.

4. Convert 723.2952 hours to the format *days : h : min : s*.

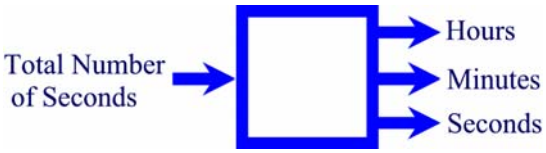
5. Write a VB program that can convert a time specified in seconds to the format *h : min : s*. Use the "Addition Calculator" program as a model of how to write code for inputting the number of seconds.

Time Converter VB Solution – Version 1

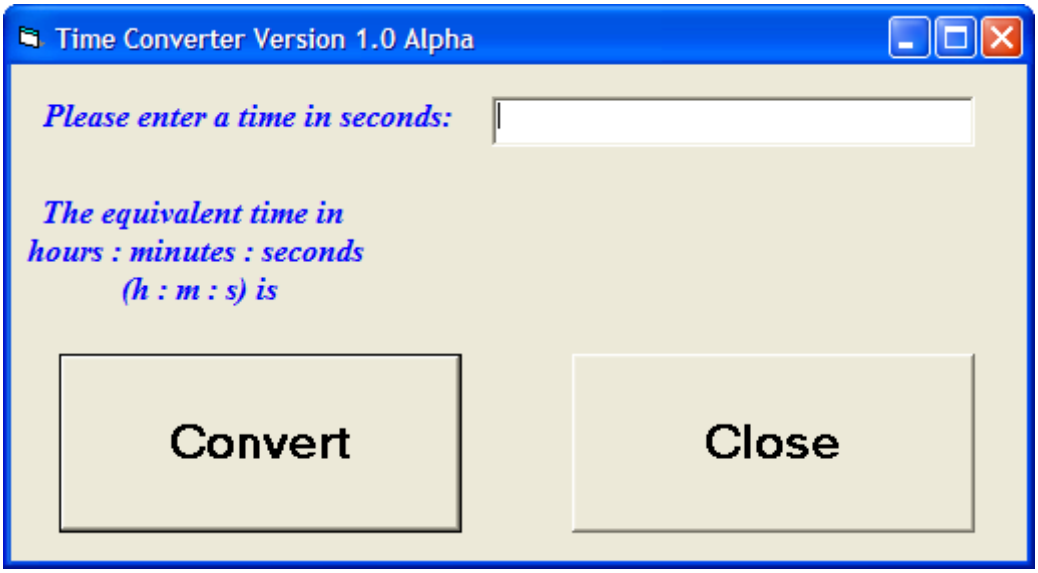
A Review of the Basic Principles of Problem Solving

George Polya's Four Steps of Problem Solving	*Corresponding Steps in Software Development (Systems Analysis)*
1. Understand the problem. 2. Choose a strategy. 3. Execute the strategy. 4. Check the solution.	1. **Analysis**: Analyze the problem and understand exactly what is required. 2. **Design**: Select algorithms and data structures. Several alternatives should be investigated. 3. **Implementation**: Write code! 4. **Validation**: Test and debug your code. 5. **Maintenance**: Release patches, updates. Plan new versions.

A Review of how we applied the above Steps to the Time Converter Problem

Analysis	We gained an understanding of the problem by reading carefully and asking questions.
Design	We worked out a ***specific example*** of the problem to gain some clues about a general strategy. We quickly learned that the ***quotient*** of integer division by 3600 (the number of seconds in one hour) is equal to the number of hours. The ***remainder*** of integer division by 3600 is equal to the remaining number of seconds. Repeating this process with integer division by 60 leads to the number of minutes and the number of seconds.  We learned about the **"Mod"** and **"\"** VB operators. (Note that the "\" is sometimes called "div.")
Implementation	We wrote the code for version 1 (see code below).
Validation	We carefully tested the program to expose any bugs or limitations. We discovered that the program worked well as long as the value entered for the total number of seconds was within the range of a "**Long**" integer variable. In addition, we learned that the program behaved strangely if a negative integer was entered.
Maintenance	This part is yet to be done. We shall soon attempt to resolve the limitations mentioned above and to add functionality to the program.

Time Converter Version One



Code for Time Converter Version 1.0 Alpha

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' PROGRAMMER'S NAME: Nick E. Nolfi                         VERSION: Time Converter Version 1.0 Alpha
'
' PURPOSE OF PROGRAM: Convert a time given in seconds to the format hours : minutes : seconds (h:m:s).
'
' LIMITATIONS and BUGS
' This program will work only if the value entered is within the range of a "Long" integer
' variable (up to 2^31 - 1).  In addition, this program will produce erroneous results if the
' user enters a negative value.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Option Explicit

Private Sub cmdClose_Click()
    Dim Response As VbMsgBoxResult
    Response = MsgBox("Are you sure you wish to close this program?", _
                      vbYesNo + vbDefaultButton2, "Leaving so soon?")
    If Response = vbYes Then
        End
    End If
End Sub
```

What is the purpose of "**Option Explicit**?"

What is the purpose of this underscore?

```
'Convert a time specified in seconds to the format hours:minutes:seconds.
Private Sub cmdConvert_Click()

    'Memory
    Dim SecondsRemaining As Long, Hours As Long, Minutes As Byte

    'Input
    SecondsRemaining = Val(txtSeconds.Text)

    'Processing
    Hours = SecondsRemaining \ 3600
    SecondsRemaining = SecondsRemaining Mod 3600
    Minutes = SecondsRemaining \ 60
    SecondsRemaining = SecondsRemaining Mod 60

    'Output
    lblHoursMinutesSeconds.Caption = CStr(Hours) & " : " & _
                                     CStr(Minutes) & " : " & _
                                     CStr(SecondsRemaining)

End Sub
```

What general term would you use to describe "cmdConvert?" What general term would you use to describe "Click?"

The name "cmdConvert_Click" is the name of the

____

Why is it sufficient to declare "Minutes" as a "**Byte**" variable? Why would it be foolish to do the same for the variables "Hours" and "SecondsRemaining?"

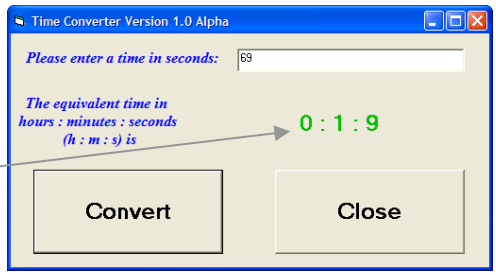
What is the "&" operator called? What is its purpose?

Extensions of this Problem

1. Find at least two different ways of preventing the user from entering a negative number. Then choose the method that you think is most user-friendly and write appropriate code.

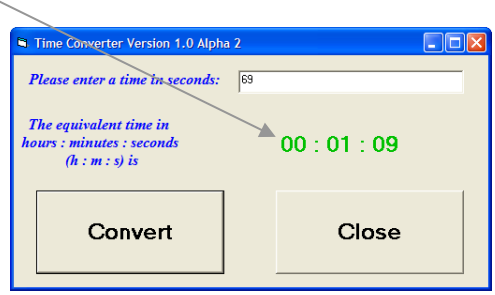
2. Suggest at least two ways of dealing with the "numeric overflow" crash caused by entering a value that exceeds the upper limit of a "**Long**" variable. Write appropriate code.

Note: Once you have completed questions 1 and 2, you will have produced a new version of the "Time Converter" program. For a complete solution, see **I:\OUT\Nolfi\Ics3m0\Time Converter Examples\Time Converter 1.0beta**.



3. Time Converter 1.0 Alpha produces messy output when the number of minutes and/or the number of seconds is less than 10. For example, if the value 69 is entered, version 1.0 alpha displays "0:1:9" instead of "0:01:09."

   The following uses "**If**" statements to solve this problem.




```
'Convert a time specified in seconds to the format hours:minutes:seconds.
Private Sub cmdConvert_Click()

    'MEMORY
    Dim SecondsRemaining As Long, Hours As Long, Minutes As Byte
    Dim SecsString As String, MinsString As String, HoursString As String

    'INPUT
    SecondsRemaining = Val(txtSeconds.Text)

    'PROCESSING
    Hours = SecondsRemaining \ 3600
    SecondsRemaining = SecondsRemaining Mod 3600
    Minutes = SecondsRemaining \ 60
    SecondsRemaining = SecondsRemaining Mod 60

    'Convert to string form
    SecsString = CStr(SecondsRemaining)
    MinsString = CStr(Minutes)
    HoursString = CStr(Hours)

    'Check if times are single digit numbers. If so, add a leading "0"
    If SecondsRemaining < 10 Then
        SecsString = "0" & SecsString
    End If

    If Minutes < 10 Then
        MinsString = "0" & MinsString
    End If

    If Hours < 10 Then
        HoursString = "0" & HoursString
    End If

    'OUTPUT
    lblHoursMinutesSeconds.Caption = HoursString & " : " & MinsString & " : " & SecsString

End Sub
```

Notice that three ***independent*** "**If**" statements are used here. The reason for this is that it is necessary to make a separate decision for each case. For instance, whether a "0" needs to be concatenated to "SecsString" depends only on the value of "SecondsRemaining." It has nothing to do with the value of "Minutes" or "Hours." Keep in mind that the "**If**…**ElseIf**…**Else**" structure should only be used when a single group of statements is selected and all the others are rejected.

Case Study 2: Storage Space and Data Transfer Rate Unit Converter Problem

In the first unit of this course, you studied storage space units, data transfer rate units and how to convert from one unit to another. The following table is a summary of all storage space and data transfer rate units. Note that the prefixes used are the same as those used for the SI system of units. However, since computer circuits are based on the binary number system, the prefix "kilo" ***usually*** stands for $1024 = 2^{10}$ instead of $1000 = 10^3$. Unfortunately, the usage of the binary meaning of "kilo" is inconsistent at best. Hardware manufacturers often use the decimal meaning, especially for data transfer rates.

Factor	*Storage Space Units*	*Data Transfer Rate Units*	
	Units Based on Bytes (binary)	Units Based on Bytes/s (binary)	Units Based on bps (decimal)
	8 b = 1 B	8 bps =1 B/s	1 bps
2^{10}	1 KB = 1024 B = 2^{10} B	1 KB/s = 1024 B/s = 2^{10} B/s	1 kbps = 1000 bps = 10^3 bps
2^{20}	1 MB = 1024 KB = 2^{20} B	1 MB/s = 1024 KB/s = 2^{20} B/s	1 Mbps = 1000000 bps= 10^6 bps
2^{30}	1 GB = 1024 MB = 2^{30} B	1 GB/s = 1024 MB/s = 2^{30} B/s	1 Gbps = 1000000000 bps= 10^9 bps
2^{40}	1 TB = 1024 GB = 2^{40} B	1 TB/s = 1024 GB/s = 2^{40} B/s	1 Tbps = 1000000000000 bps= 10^{12} bps
2^{50}	1 PB = 1024 TB = 2^{50} B	1 PB/s = 1024 TB/s = 2^{50} B/s	1 Pbps = 10^{15} bps
2^{60}	1 EB = 1024 PB = 2^{60} B	1 EB/s = 1024 PB/s = 2^{60} B/s	1 Ebps = 10^{18} bps
2^{70}	1 ZB = 1024 EB = 2^{70} B	1 ZB/s = 1024 EB/s = 2^{70} B/s	1 Zbps = 10^{21} bps
2^{80}	1 YB = 1024 ZB = 2^{80} B	1 YB/s = 1024 ZB/s = 2^{80} B/s	1 Ybps = 10^{24} bps

Note

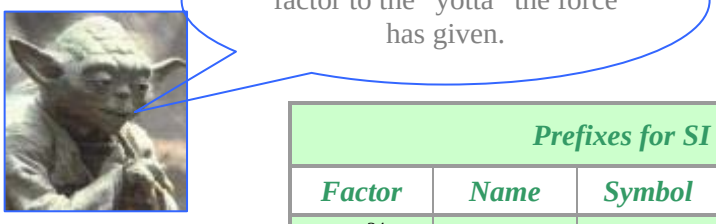
1. For Storage Space Units "Kilo" means $1024 = 2^{10}$

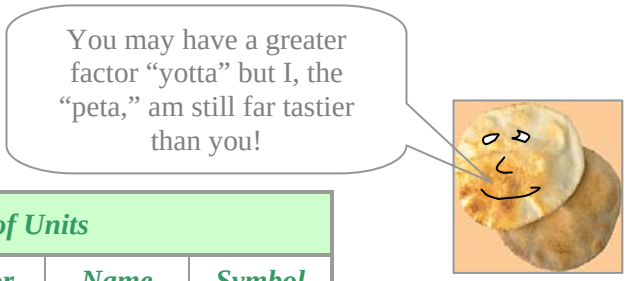
The prefix "kilo" usually means $1000 = 10^3$, but since computers are based on "twos" (binary), a power of 2 is much more convenient than a power of 10. The value 1024 was chosen because it is the power of 2 closest to 1000.

2. Ambiguous use of "Kilo" for Storage Capacity and Data Transfer Rate Units

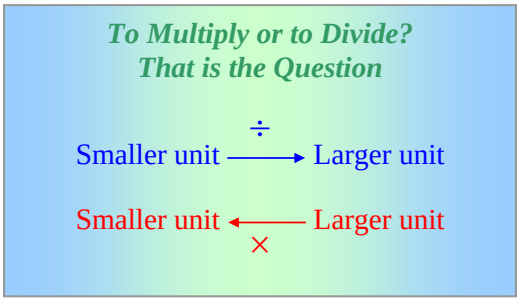
Despite the point made in "1," hardware manufacturers very often use the decimal (SI) meaning of "kilo," especially for data transfer rates. In addition, in the SI system of units, the prefix lowercase "k" is used for "kilo." When dealing with storage capacity and data transfer rate units, however, both uppercase "K" and lowercase "k" can be used. By convention, uppercase "K" means 1024 while lowercase "k" means 1000. Thus 1 KB = 1024 B while 1 kB = 1000 B. (Unfortunately, even this convention is not used consistently.)

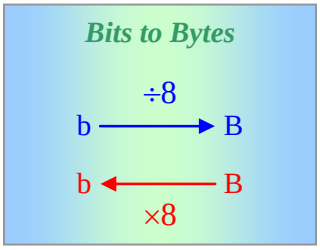
The following table summarizes the prefixes for the SI system of units (decimal, not binary).



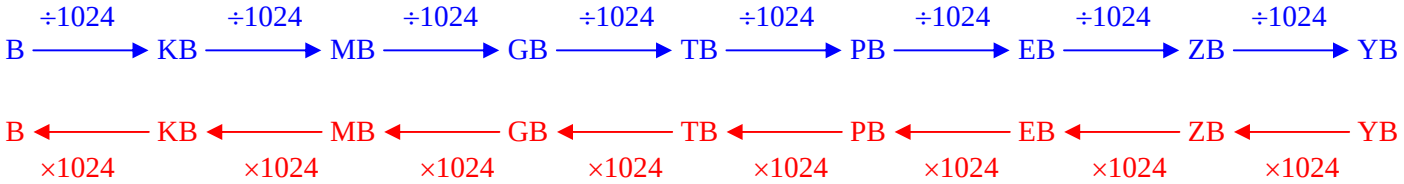


Prefixes for SI System of Units					
Factor	*Name*	*Symbol*	*Factor*	*Name*	*Symbol*
10^{24}	yotta	Y	10^{-1}	deci	d
10^{21}	zetta	Z	10^{-2}	centi	c
10^{18}	exa	E	10^{-3}	milli	m
10^{15}	peta	P	10^{-6}	micro	μ
10^{12}	tera	T	10^{-9}	nano	n
10^{9}	giga	G	10^{-12}	pico	p
10^{6}	mega	M	10^{-15}	femto	f
10^{3}	kilo	k	10^{-18}	atto	a
10^{2}	hecto	h	10^{-21}	zepto	z
10^{1}	deka	da	10^{-24}	yocto	y

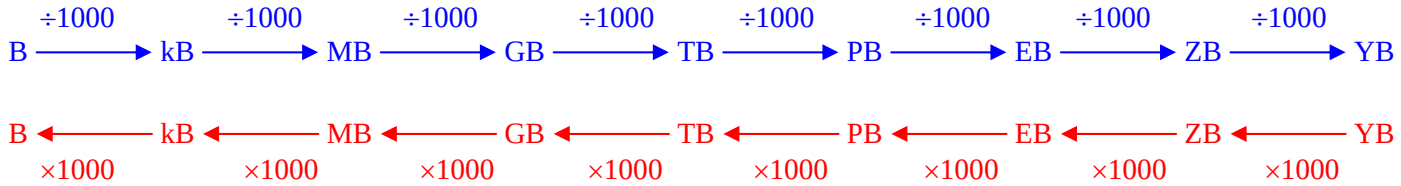




Conversion Table (for Kilo=1024)



Conversion Table (for kilo=1000)



Exercises

1. The *Rogers Yahoo! Hi-Speed Internet Extreme* service has a maximum downstream data transfer rate (download speed) of 7 Mbps and a maximum upstream data transfer rate of (upload speed) 512 kbps. (Note that for these two rates, $\text{kilo} = 1000 = 10^3$ and $\text{mega} = 1000000 = 10^6$.)

(a) Convert the downstream data rate from Mbps (megabits per second, M=1000000) to KB/s (kilobytes per second, K=1024).

(b) Convert the upstream data rate from kbps (k=1000) to KB/s (K=1024).

(c) How long would it take to ***download*** (receive) the administrative version of Windows XP Service Pack 2 (272391 KB)? Assume that the data can be transferred at the maximum rate of 6 Mbps. State your answer in hours, minutes and seconds.

(d) When you use a bit-torrent client such as Azureus, your computer becomes connected to what is known as a peer-to-peer (P2P) file sharing network. As you download (receive) files from other users, your computer also uploads (sends) files. How long would it take to ***upload*** (send) the administrative version of Windows XP Service Pack 3 (324030 KB) to another user? Assume that the data can be transferred at the maximum rate of 512 kbps. State your answer in hours, minutes and seconds.

2. Suppose that you had a 500 GB hard drive that you wanted to back up. How many of each of the following storage media would you need to use, assuming that there is no free space on the hard drive.

(a) 1.44 MB floppy diskettes

(b) 700 MB CD-R disks

A Proposal to Avoid the Confusion caused by two Possible Meanings of "Kilo"

Introduction

Knowing whether "kilo" refers to 1000 or 1024 can cause a great deal of confusion. To prevent this confusion, a new set of prefixes has been introduced. Information about these prefixes from ***three different Web sites*** is given below. Read all the information and then answer the questions at the bottom of the page.

A Description of "Kibibyte" from Wikipedia

Kibibyte

From Wikipedia, the free encyclopedia

A **kibibyte** (a contraction of **ki**lo **bi**nary **byte**) is a unit of information or computer storage, abbreviated **KiB** (never "kiB").

1 kibibyte = 2^{10} bytes = 1,024 bytes

The kibibyte is closely related to the kilobyte, which can be used either as a synonym for kibibyte or to refer to 10^3 bytes = 1,000 bytes (see binary prefix).

Usage of these terms is intended to help prevent the confusion common among storage media, due to the ambiguous meaning of "kilobyte". Thus the term **kibibyte** has evolved to refer exclusively to 1,024 bytes.

This problem of confusion of the term *kilobyte* simultaneously being used to refer to both 1,000 and 1,024 became more prevalent when computer hard drives grew to the gigabyte and larger size, because if one expects power of two values to refer to capacity, and manufacturers were using power of ten values, the difference could be substantial, e.g. 1 megabyte, if expressed as power of two, is 1024^2 or 1024×1024, or 1,048,576, while 1000×1000 is 1,000,000. In the case of a "gigabyte", if one uses 1024^3, the size of a drive would be expected to be 1,073,741,824 bytes per gigabyte versus 1000^3, or a mere 1,000,000,000. On a 100 gigabyte drive, the difference is more than 7 billion characters additional storage, depending on whether 100 gigabytes refers to 100×1000^3 or 100×1024^3.

Quantities of bytes				
SI prefixes			Binary prefixes (IEC 60027-2)	
Name (Symbol)	Popular Usage	Standard SI	Name (Symbol)	Value
kilobyte (kB)	2^{10}	10^3	**kibibyte** (KiB)	2^{10}
megabyte (MB)	2^{20}	10^6	mebibyte (MiB)	2^{20}
gigabyte (GB)	2^{30}	10^9	gibibyte (GiB)	2^{30}
terabyte (TB)	2^{40}	10^{12}	tebibyte (TiB)	2^{40}
petabyte (PB)	2^{50}	10^{15}	pebibyte (PiB)	2^{50}
exabyte (EB)	2^{60}	10^{18}	exbibyte (EiB)	2^{60}
zettabyte (ZB)	2^{70}	10^{21}	zebibyte (ZiB)	2^{70}
yottabyte (YB)	2^{80}	10^{24}	yobibyte (YiB)	2^{80}

A Description of "Kibibyte" from FOLDOC

The official ISO[?] name for 1024 bytes, to distinguish it from 1000 bytes which they call a kilobyte. "Mebibyte," "Gibibyte," etc, are prefixes for other powers of 1024. Although this new naming standard has been widely reported in 2003, it seems unlikely to catch on.

A Description of "Kibibyte" from http://www.robinlionheart.com/stds/html4/glossary

kibibyte (KiB)
A ***kibibyte*** is a unit of storage equal to exactly 1,024 bytes. Because kilobyte is used to mean either 1000 bytes or 1024 bytes, in 1999 the International Electrotechnical Commission defined a "kibi-" prefix unambiguously signifying 1024. Rarely used except by pedantic nerds, like me.

Questions

1. Explain why "kilo=1000" is called the ***decimal*** meaning and "kilo=1024" is called the ***binary*** meaning.
2. Define the words ***pedantic***, ***nerd***, ***ambiguous***, ***standard*** and ***convention***.
3. Since 1000 is very close to 1024, why should anyone bother distinguishing between the two meanings of "kilo?"
4. When using the Internet to do research, do you think that it would be wise to consult only one Web site? Explain.
5. Are there any inconsistencies in the three sources of information?
6. The manufacturers of two different hard drives both claim that the storage capacity of the drives is 1 TB. One manufacturer uses the "kilo=1024" definition and the other uses the "kilo=1000" definition. Calculate the difference in storage capacities between the two drives.

Problems that can be Solved by Investigating Specific Examples

1. Convert a time specified in seconds to the form *hours*:*minutes*:*seconds*. (**e.g.** 3642 s = 1 h : 0 min : 42 s)
2. Convert a time specified in minutes to the form *hours*:*minutes*:*seconds*. (**e.g.** 125.6 min = 2 h : 5 min : 36 s)
3. Convert a time specified in hours to the form *hours*:*minutes*:*seconds*. (**e.g.** 25.66 h = 25 h : 39 min : 36 s)
4. Convert any time specified in *days*:*hours*:*minutes*:*seconds* to the best possible form in *days*:*hours*:*minutes*:*seconds*. (**e.g.** 2 days : 63 h : 189 min : 322 s = 4 days : 18 h : 14 min : 22 s)
5. Convert a certain amount of money to the form "# $1000 bills, # $100 bills, # $50 bills, # $20 bills, # $10 bills, # $5 bills, # $2 coins, # $1 coins, # $0.25 coins, # $0.10 coins, # $0.05 coins, # $0.01 coins" (**e.g.** $7987.32 = **seven** $1000 bills, **nine** $100 bills, **one** $50 bill, **one** $20 bill, **one** $10 bill, **one** $5 bill, **one** $2 coin, **zero** $1 coins, **one** $0.25 coin, **zero** $0.10 coins, **one** $0.05 coins, **two** $0.01 coins)
6. Given any two fractions, add them, subtract them, multiply them or divide them.
7. Convert any storage capacity unit into any other.
8. Convert any data transfer rate unit into any other.

Assignment

1. Solve a specific example and write an algorithm for each of the eight problems listed above. Arrange your work in table format as shown below. An example is given to help you understand what is required.

Specific Example	*Algorithm*
Convert 35356 s to the format *h : min : s*. **hours** | **minutes** | **seconds** Step 1: 0 | 0 | 35356 Step 2: 9 | 0 | 2956 Step 3: 9 | 49 | 16 35356 ÷ 3600 = 9 R 2956, 2956 ÷ 60 = 49 R 16	1. The user enters a time in seconds: ***seconds*** 2. Set ***hours*** to the quotient of ***seconds*** divided by 3600 3. Set ***seconds*** to the remainder of ***seconds*** divided by 3600 4. Set ***minutes*** to the quotient of ***seconds*** divided by 60 5. Set ***seconds*** to the remainder of ***seconds*** divided by 60 6. The result is ***hours*** : ***minutes*** : ***seconds***

2. Create a VB program that
   - **(a)** can convert any data storage capacity unit into any other
   - **(b)** can convert any data transfer rate unit into any other
   - **(c)** allows the user to use either the binary or decimal meaning of "kilo"
     binary: base 2, Kilo = K = $1024 = 2^{10}$ **decimal:** base 10, kilo = k = $1000 = 10^3$

Evaluation Guide for Question 1

Categories	*Criteria*	*Descriptors*					*Level*	*Average*
		Level 4	Level 3	Level 2	Level 1	Level 0		
Knowledge and Understanding (KU)	**Understanding of the Problems**	Extensive	Good	Moderate	Minimal	Insufficient		
Application (APP)	**Correctness of Chosen Examples** To what degree are the chosen examples solved correctly?	Very High	High	Moderate	Minimal	Insufficient		
Thinking, Inquiry and Problem Solving (TIPS)	**Appropriateness of Chosen Examples** To what degree has the student chosen non-trivial examples that can be extended to general algorithms?	Very High	High	Moderate	Minimal	Insufficient		
	Generality of Algorithms To what degree are the algorithms applicable to the given problems?	Very High	High	Moderate	Minimal	Insufficient		
Communication (COM)	**Clarity of Solutions of Chosen Examples** How clearly are the solutions of the chosen examples communicated?	Extremely Easy to Under-stand	Easy to Under-stand	Moderately Easy to Under-stand	Somewhat Abstruse	Extremely Abstruse		
	Clarity of Algorithm Descriptions How clearly are the algorithms communicated?	Extremely Easy to Under-stand	Easy to Under-stand	Moderately Easy to Under-stand	Somewhat Abstruse	Extremely Abstruse		

Evaluation Guide for Question 2 (Unit Conversion Program)

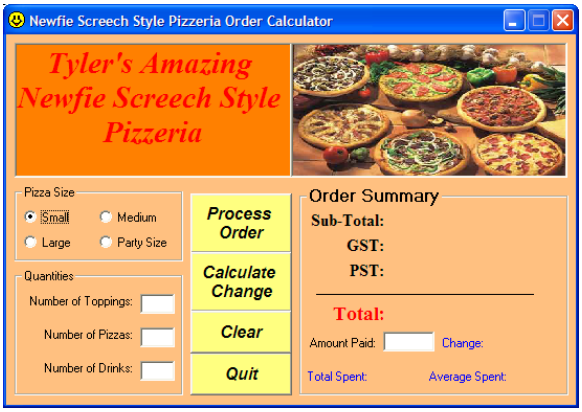
Categories	*Criteria*	*Descriptors*					*Level*	*Average*
		Level 4	Level 3	Level 2	Level 1	Level 0		
Knowledge and Understanding (KU)	**Understanding of Programming Concepts**	Extensive	Good	Moderate	Minimal	Insufficient		
	Understanding of the Problem	Extensive	Good	Moderate	Minimal	Insufficient		
Application (APP)	**Correctness** To what degree is the output correct?	Very High	High	Moderate	Minimal	Insufficient		
	Declaration of Variables To what degree are the variables declared with appropriate data types?	Very High	High	Moderate	Minimal	Insufficient		
	Debugging To what degree has the student employed a logical, thorough and organized debugging method?	Very High	High	Moderate	Minimal	Insufficient		
Thinking, Inquiry and Problem Solving (TIPS)	**Algorithm Design and Selection** To what degree has the student used approaches such as solving a specific example of the problem to gain insight into the problem that needs to be solved?	Very High	High	Moderate	Minimal	Insufficient		
	Ability to Design and Select Algorithms Independently To what degree has the student been able to design and select algorithms without assistance?	Very High	High	Moderate	Minimal	Insufficient		
	Ability to Implement Algorithms Independently To what degree is the student able to implement chosen algorithms without assistance?	Very High	High	Moderate	Minimal	Insufficient		
	Efficiency of Algorithms and Implementation To what degree does the algorithm use resources (memory, processor time, etc) efficiently?	Very High	High	Moderate	Minimal	Insufficient		
Communication (COM)	**Indentation of Code** **Insertion of Blank Lines in Strategic Places** (to make code easier to read)	Very Few or no Errors	A Few Minor Errors	Moderate Number of Errors	Large Number of Errors	Very Large Number of Errors		
	Comments • Effectiveness of explaining abstruse (difficult-to-understand) code • Effectiveness of introducing major blocks of code • Avoidance of comments for self-explanatory code	Very High	High	Moderate	Minimal	Insufficient		
	Descriptiveness of Identifier Names Variables, Constants, Objects, Functions, Subs, etc **Inclusion of Property Names with Object Names** (**e.g.** 'txtName.Text' instead of 'txtName' alone) **Clarity of Code** How easy is it to understand, modify and debug the code? **Adherence to Naming Conventions** (**e.g.** use "txt" for text boxes, "lbl" for labels, etc.)	Masterful	Good	Adequate	Passable	Insufficient		
	User Interface To what degree is the user interface well designed, logical, attractive and user-friendly?	Very High	High	Moderate	Minimal	Insufficient		

Problem Solving Strategy 2: Plan your Solution in a Logical, Organized Fashion

The Problem that you need to Solve

Since Tyler is so busy kneading the dough for his Newfie Screech Style Pizza, he does not have much time to process customer orders. Therefore, he is seeking your help! His restaurant, ***Newfie Screech Style Pizzeria***, needs a computer program that can process customer orders.

As shown in the table, there is a base price for each pizza, plus an additional charge for each topping.



Size	Base Price	Each Topping
Small	$9.95	$1.00
Medium	$12.95	$1.25
Large	$15.95	$1.50
Party Size	$18.95	$2.00
Drinks	$1.25	

Write a Visual Basic program that uses the form shown below to

(1) Input the ***size*** of the pizza, the ***number of toppings***, the ***number of pizzas*** and the number of ***drinks***

(2) Calculate and display the ***sub-total*** (cost before tax), the ***PST*** (8%), the ***GST*** (6%) and the ***total***

(3) Input the ***amount of money paid*** by the customer

(4) Calculate and display the ***change*** that the customer should receive

(5) Calculate and display the ***total*** amount spent by all customers

(6) Calculate and display the ***average*** amount spent by each customer.

Local Variables versus Global Variables

Local Variables	*Global Variables*
As shown below, local variables are declared ***inside* Subs**. `Private Sub cmdCalculateChange_Click()` `Dim Change As Currency, CashTendered As Currency` Local variables are 1. VISIBLE only within the sub in which they are declared. 2. CREATED when the sub is invoked (i.e. called or executed). 3. DESTROYED when the sub returns (has finished executing). Local variables should be used whenever possible. They help to reduce the time needed to debug a program because they keep information PRIVATE. If information is needed only by a particular sub, it is best to HIDE it from other subs. Local variables also help to conserve memory because they are discarded as soon as the sub returns.	As shown below, local variables are declared ***at the top of the code***, just after **Option Explicit**. `Option Explicit` `Dim TotalCostOfOrder As Currency` 1. The values of global variables remain stored in RAM as long as the form is loaded in RAM (i.e. the computer will "remember" the values of these variables for as long as the form remains loaded 2. Global variables are VISIBLE to all the subs. Each sub can access each global variable, allowing two or more subs to SHARE their values. A variable should be declared GLOBALLY whenever two or more subs need to access it (i.e. use or change its value) and/or whenever its value needs to be "remembered" after a sub has finished executing.

The Plan

INPUT	PROCESSING	OUTPUT
What information does the user enter?	What must be done with the information?	What should be displayed after processing is complete?
Code for Input	**Code for Processing**	**Code for Output**
VARIABLES (MEMORY)		
LOCAL VARIABLES		**GLOBAL VARIABLES**

Pizza Program Solutions and Questions

SIZE	BASE PRICE	EACH TOPPING
Small	$9.95	$1.00
Medium	$12.95	$1.25
Large	$15.95	$1.50
Party Size	$18.95	$2.00
Drinks	$1.25	

The Problem

"Newfoundland Style Pizzeria Problem"

The Plan

INPUT

What information must the user enter?

Process Order Button

Pizza Size, Number of Pizzas, Number of Toppings, Number of Drinks

Calculate Change Button

Amount of money customer pays.

PROCESSING

What must be done with the information?

Process Order Button

1. Determine base price for pizza size chosen
2. Determine price per topping for chosen size
3. Calculate cost before taxes (subtotal):
   #pizzas*[(base price) + #toppings*(topping price)] + #drinks*(drink price)
4. Calculate GST and PST
   GST: subtotal*0.06, PST: subtotal*0.08
5. Calculate total for order: Subtotal + GST + PST
6. Add (order total) to (total for all customers)
7. Increase the number of orders by 1
8. Calculate the average cost of each order: (total spent by all) / (#orders)

Calculate Change Button

Calculate change.

OUTPUT

What should be displayed after processing is complete?

Process Order Button

1. Display subtotal
2. Display GST
3. Display PST
4. Display total
5. Display total spent by all customers
6. Display average amount spent by each customer

Calculate Change Button

Display change.

2. Explain the purpose of the "NumOrders" variable.

1. Explain why ***most*** of the variables are declared as ***local variables*** while a few are declared as ***global variables***.

VARIABLES (MEMORY)

LOCAL VARIABLES

Integer Variables

NumPizzas
NumToppings
NumDrinks

These variables store values that involve a number of items

Currency Variables

PizzaBasePrice

PricePerTopping, SubTotal, GST, PST

Change

CashTendered

AverageAmountSpent

These variables store values that involve an amount of money

GLOBAL VARIABLES

Currency Variables

TotalCostOfOrder

TotalSpentByAllCustomers

NumOrders

The Code

A complete VB solution for this problem can be found in the folder

I:\OUT\Nolfi\Ics3m0\Simple VB Examples\Newfie Pizza Example

Only the global variables and the "`cmdProcessOrder_Click()`" sub are shown here.

```
Option Explicit 'Used to force variable declarations.

'GLOBAL VARIABLES
Dim TotalCostOfOrder As Currency, TotalSpentByAllCustomers As Currency
Dim NumOrders As Integer
Private Sub cmdProcessOrder_Click()

    'MEMORY: LOCAL VARIABLES
    Dim PizzaBasePrice As Currency, PricePerTopping As Currency, SubTotal As Currency
    Dim GST As Currency, PST As Currency, AverageAmountSpent As Currency
    Dim NumPizzas As Integer, NumDrinks As Integer, NumToppings As Integer

    'INPUT: Obtain information from user.
    NumPizzas = Val(txtPizzas.Text)
    NumToppings = Val(txtToppings.Text)
    NumDrinks = Val(txtDrinks.Text)

    'PROCESSING
    'Decide what the base price and price per topping should be.
    If optSmall.Value = True Then
        PizzaBasePrice = 9.95
        PricePerTopping = 1
    ElseIf optMedium.Value = True Then
        PizzaBasePrice = 12.95
        PricePerTopping = 1.25
    ElseIf optLarge.Value = True Then
        PizzaBasePrice = 15.95
        PricePerTopping = 1.5
    Else
        PizzaBasePrice = 18.95
        PricePerTopping = 2
    End If

    'Now perform all calculations
    SubTotal = (PizzaBasePrice + PricePerTopping * NumToppings) _
                                                * NumPizzas + NumDrinks * 1.25
    GST = Round(SubTotal * 0.07, 2)
    PST = Round(SubTotal * 0.08, 2)
    TotalCostOfOrder = SubTotal + GST + PST
    TotalSpentByAllCustomers = TotalSpentByAllCustomers + TotalCostOfOrder
    NumOrders = NumOrders + 1
    AverageAmountSpent = Round(TotalSpentByAllCustomers / NumOrders, 2)

    'OUTPUT: Display results.
    lblSubTotal.Caption = Format(SubTotal, "Currency")
    lblGST.Caption = Format(GST, "Currency")
    lblPST.Caption = Format(PST, "Currency")
    lblTotal.Caption = Format(TotalCostOfOrder, "Currency")
    lblTotalSpent.Caption = Format(TotalSpentByAllCustomers, "Currency")
    lblAverageSpent.Caption = Format(AverageAmountSpent, "Currency")

End Sub
```

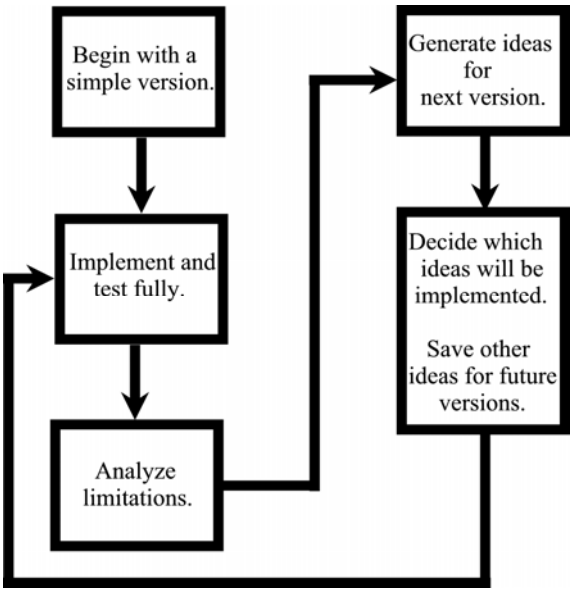
Questions

1. If all variables in this program were declared locally, would this program still work correctly? Explain.
2. If all variables in this program were declared globally, would this program still work correctly? If so, is it a good idea to declare all variables globally? Explain.
3. Explain the purpose of the intrinsic functions "Round" and "Format." Use MSDN help to find technical information on these two functions.

Problem Solving Strategy 3: Break up Large, Complex Problems into a Series of Smaller, Simpler Problems


(Don't be a Pumpkin head! Follow these Guidelines!)


The Infinite Loop of Software Development



The flowchart shown at the left is a simplified visual representation of the software development process. Notice that programmers use a simple version as a foundation upon which future versions can be built. Also, note that once the initial simple version has been implemented, an essentially infinite loop is entered. Since software development involves open-ended tasks, there is virtually no limit to the improvements that can be made!
When engaged in this process, try to keep in mind the following points:

- Break up large, complex problems into several smaller problems.
- Solve one small problem at a time. Ensure that each solution is perfect before integrating it into the overall system.
- Be realistic! It is far better to produce simple software that works well than it is to produce sophisticated software that does not work at all.
- Do not limit yourself during the idea generation phase. Write down all your ideas (including those that seem over-ambitious or downright crazy).

Some General Guidelines for Producing Great Code

- Use names like ***InsertionPoint*** instead of *insertionpoint*, *INSERTIONPOINT*, *insertion_point* or *INSERTION_POINT*
- Use names that ***clearly describe the purpose of a variable, constant, sub procedure or function procedure.***
- Using ***meaningful, descriptive*** names will allow you to write programs that are for the most part self-explanatory. This means that you do not need to include too many comments. However, ***comments should still be considered an integral part of the software development process***. Comments should be included as you write your code, not after it is written!
- Generally, include ***comments for major blocks of code*** and for any ***code that is not self-explanatory***.
- Use global variables only when necessary! All other variables should be declared either within procedures or as parameters of procedures.
- ***Avoid repetitive code*** by writing sub procedures or function procedures and calling them whenever they are needed.
- ***Consider several different algorithms*** and implement the one that best suits your needs.
- ***Indent your code properly*** as you write it! Do not consider indentation an afterthought.
- ***Test your code thoroughly under extreme conditions***. Allow other people to conduct some of the testing and note all bugs.

The Fraction Calculator Program

Instructions

Read the memo given below. ***Before diving right into the VB code, take some time to PLAN your solution!***

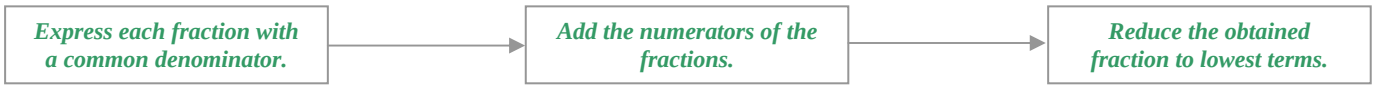
INTERNAL MEMO

From: I. M. De Boss
To: U. R. Not De Boss
Re: The **"Fraction Calculator"** software.

The "Fraction Calculator" must be able to add, subtract, multiply and divide any two fractions expressed in improper form. All answers must be displayed in lowest terms. The following is an example of the type of question that your calculator should be able to handle. Note that your calculator will only display the question and the final answer. The intermediate steps do not need to be displayed but we may tackle this in a future version.

Detailed Solution	Explanation
$\frac{6}{8}+\frac{5}{6}$	These two fractions must be added.
$=\frac{6\times3}{8\times3}+\frac{5\times4}{6\times4}$	The LCD (lowest common denominator) is 24. Express each fraction with a denominator of 24. The LCD is the ***least common multiple of 8 and 6***.
$=\frac{18}{24}+\frac{20}{24}$	Now that both fractions have the same denominator, the numerators can be added.
$=\frac{38\div2}{24\div2}$	This fraction is not reduced to lowest terms, so one more step is necessary. Bo the numerator and denominator are divided by the ***greatest common divisor of 38 and 24.***
$=\frac{19}{12}$	This is the final answer reduced to lowest terms.

Overall Plan



Pseudo-Code

1. The user enters the numerators and denominators of each fraction: ***numer1***, ***denom1***, ***numer2***, ***denom2***
2. Set ***denomAnswer*** to the least common multiple of ***denom1*** and ***denom2***
3. Set ***numer1*** to ***denomAnswer*** divided by ***denom1*** multiplied by ***numer1***
4. Set ***numer2*** to ***denomAnswer*** divided by ***denom2*** multiplied by ***numer2***
5. Set ***numerAnswer*** to ***numer1*** plus ***numer2***
6. Set ***gcd*** to the greatest common divisor of ***numerAnswer*** and ***denomAnswer***
7. Set ***numerAnswer*** to ***numerAnswer*** divided by ***gcd***
8. Set ***denomAnswer*** to ***denomAnswer*** divided by ***gcd***
9. The result is $\frac{numAnswer}{denomAnswer}$

Above Example done using Memory Map

	numer1	*denom1*	*numer2*	*denom2*	*denomAnswer*	*numerAnswer*	*gcd*
1.	**6**	**8**	**5**	**6**	-	-	-
2.	6	8	5	6	**24**	-	-
3.	**18**	8	5	6	24	-	-
4.	18	8	**20**	6	24	-	-
5.	18	8	20	6	24	**38**	-
6.	18	8	20	6	24	38	**2**
7.	18	8	20	6	24	**19**	2
8.	18	8	20	6	**12**	19	2
9.	18	8	20	6	**12**	**19**	2

Using the Fraction Calculator Assignment to Learn How to Improve Existing Code (Part 1)

Instructions

Carefully study the code shown on the next page (Fraction Calculator Version 1.00). After you do so, run the "Fraction Calculator 1.00" VB program, which you will find stored in

I:\Out\Nolfi\Ics3m0\Fraction Calculator\Fraction 1.00\Fraction 1.00.vbp

Then complete the following table.

How does the "cmdAdd_Click" sub differ from the "cmdSubtract_Click" sub?	
How does the "cmdMultiply_Click" sub differ from the "cmdDivide_Click" sub?	
Is there any code that is repeated in several different places?	
State several ways in which the code and the user interface could be improved.	

```
'Fraction Calculator Version 1.00
Private Sub cmdAdd_Click()
    Dim PossibleMulti As Long, LCM As Long
    Dim Smaller As Long, Larger As Long
    Dim Denom1 As Long, Denom2 As Long
    Dim Numerator1 As Long, Numerator2 As Long
    Dim AnsNumerator As Long, AnsDenom As Long
    Dim PossibleDivisor As Long, GCD As Long
     'Input
    Denom1 = Val(txtDenom1.Text)
    Denom2 = Val(txtDenom2.Text)
    Numerator1 = Val(txtNumer1.Text)
    Numerator2 = Val(txtNumer2.Text)
    If Denom1 < Denom2 Then
        Larger = Denom1
    Else
        Larger = Denom2
    End If
    'Find LCM of "Denom1" and "Denom2"
    For PossibleMulti = Larger To Denom1 * Denom2
        If PossibleMulti Mod Denom1 = 0 And _
                    PossibleMulti Mod Denom2 = 0 Then
            LCM = PossibleMulti
            Exit For
        End If
    Next PossibleMulti
    AnsNumerator = LCM / Denom1 * Numerator1 + LCM / _
                                    Denom2 * Numerator2
    AnsDenom = LCM
    If AnsDenom < AnsNumerator Then
        Smaller = AnsDenom
    Else
        Smaller = AnsNumerator
    End If
    'Find the GCD of "AnsNumerator" and "AnsDenom"
    GCD = 1
    For PossibleDivisor = Smaller To 2 Step -1
        If AnsNumerator Mod PossibleDivisor = 0 And _
                AnsDenom Mod PossibleDivisor = 0 Then
            GCD = PossibleDivisor
            Exit For
        End If
    Next PossibleDivisor
    'OutPut
    txtAnsNumer.Text = AnsNumerator / GCD
    txtAnsDenom.Text = AnsDenom / GCD
End Sub
Private Sub cmdMultiply_Click()
    Dim Denom1 As Long, Denom2 As Long
    Dim Numerator1 As Long, Numerator2 As Long
    Dim PossibleDivisor As Long, GCD As Long
    Dim Smaller As Long
    'Input
    Denom1 = Val(txtDenom1.Text)
    Denom2 = Val(txtDenom2.Text)
    Numerator1 = Val(txtNumer1.Text)
    Numerator2 = Val(txtNumer2.Text)
    'Processing
    AnsNumerator = Numerator1 * Numerator2
    AnsDenom = Denom1 * Denom2
    'Find the GCD of "AnsNumerator" and "AnsDenom"
    If AnsDenom < AnsNumerator Then
        Smaller = AnsDenom
    Else
        Smaller = AnsNumerator
    End If
    GCD = 1
    For PossibleDivisor = Smaller To 2 Step -1
        If AnsNumerator Mod PossibleDivisor = 0 And _
                AnsDenom Mod PossibleDivisor = 0 Then
            GCD = PossibleDivisor
            Exit For
        End If
    Next PossibleDivisor
    'OutPut
    txtAnsNumer.Text = AnsNumerator / GCD
    txtAnsDenom.Text = AnsDenom / GCD
End Sub
Private Sub cmdSubtract_Click()
    Dim PossibleMulti As Long, LCM As Long
    Dim Smaller As Long, Larger As Long
    Dim Denom1 As Long, Denom2 As Long
    Dim Numerator1 As Long, Numerator2 As Long
    Dim AnsNumerator As Long, AnsDenom As Long
    Dim PossibleDivisor As Long, GCD As Long
    'Input
    Denom1 = Val(txtDenom1.Text)
    Denom2 = Val(txtDenom2.Text)
    Numerator1 = Val(txtNumer1.Text)
    Numerator2 = Val(txtNumer2.Text)
    If Denom1 < Denom2 Then
        Larger = Denom1
    Else
        Larger = Denom2
    End If
    'Find LCM of "Denom1" and "Denom2"
    For PossibleMulti = Smaller To Denom1 * Denom2
        If PossibleMulti Mod Denom1 = 0 And _
                    PossibleMulti Mod Denom2 = 0 Then
            LCM = PossibleMulti
            Exit For
        End If
    Next PossibleMulti
    AnsNumerator = LCM / Denom1 * Numerator1 - LCM / _
                                    Denom2 * Numerator2
    AnsDenom = LCM
    If AnsDenom < AnsNumerator Then
        Smaller = AnsDenom
    Else
        Smaller = AnsNumerator
    End If
    'Find the GCD of "AnsNumerator" and "AnsDenom"
    GCD = 1
    For PossibleDivisor = Smaller To 2 Step -1
        If AnsNumerator Mod PossibleDivisor = 0 And _
                AnsDenom Mod PossibleDivisor = 0 Then
            GCD = PossibleDivisor
            Exit For
        End If
    Next PossibleDivisor
    'OutPut
    txtAnsNumer.Text = AnsNumerator / GCD
    txtAnsDenom.Text = AnsDenom / GCD
End Sub
Private Sub cmdDivide_Click()
    Dim Denom1 As Long, Denom2 As Long
    Dim Numerator1 As Long, Numerator2 As Long
    Dim PossibleDivisor As Long, GCD As Long
    Dim Smaller As Long
    'Input
    Denom1 = Val(txtDenom1.Text)
    Denom2 = Val(txtDenom2.Text)
    Numerator1 = Val(txtNumer1.Text)
    Numerator2 = Val(txtNumer2.Text)
    'Processing
    AnsNumerator = Numerator1 * Denom2
    AnsDenom = Denom1 * Numerator2
    'Find the GCD of "AnsNumerator" and "AnsDenom"
    If AnsDenom < AnsNumerator Then
        Smaller = AnsDenom
    Else
        Smaller = AnsNumerator
    End If
    GCD = 1
    For PossibleDivisor = Smaller To 2 Step -1
        If AnsNumerator Mod PossibleDivisor = 0 And _
                AnsDenom Mod PossibleDivisor = 0 Then
            GCD = PossibleDivisor
            Exit For
        End If
    Next PossibleDivisor
    'Output
    txtAnsNumer.Text = AnsNumerator / GCD
    txtAnsDenom.Text = AnsDenom / GCD
End Sub
```

Using the Fraction Calculator Assignment to Learn How to Improve Existing Code (Part 2)

Instructions

Carefully study the code shown on the next page (Fraction Calculator Version 1.01). After you do so, run the "Fraction Calculator 1.01" VB program, which you will find stored in

I:\Out\Nolfi\Ics3m0\Fraction Calculator\ Fraction 1.01\Fraction 1.01.vbp

Then complete the following table.

How does version 1.01 differ from version 1.00?	
`` Dim Denom1 As Long, Denom2 As Long Dim Numerator1 As Long, _ Numerator2 As Long ``` In version 1.00, the above declarations appeared within the subs (i.e. the variables were declared as ***local variables***). Why is it necessary to declare these variables ***globally*** in version 1.01?	
Use MSDN help or any other resources to do research on the following topics: • Function Procedures • Sub Procedures • General Sub Procedures versus Event Sub Procedures • Passing Parameters (Arguments) "By Value" • Passing Parameters (Arguments) "By Reference"	

```
'Fraction Calculator Version 1.01
'The code for this version is considerably shorter
'than the code for version 1.00.  This is due to the
'use of "Sub Procedures" and "Function Procedures."
Option Explicit

Dim Denom1 As Long, Denom2 As Long
Dim Numerator1 As Long, Numerator2 As Long

Private Sub cmdAdd_Click()
    'Memory
    Dim AnsNumerator As Long, AnsDenom As Long
    Dim GCD As Long, PossibleMulti As Long, LCM As Long
    'Input
    Call GetInput
    'Processing
    LCM = LeastCommonMultiple(Denom1, Denom2)
    AnsNumerator = LCM / Denom1 * Numerator1 + LCM / _
                                    Denom2 * Numerator2
    AnsDenom = LCM
    GCD = GreatestCommonDivisor(AnsNumerator, AnsDenom)
    'Output - Display answer as a reduced fraction
    txtAnsNumer.Text = AnsNumerator / GCD
    txtAnsDenom.Text = AnsDenom / GCD
    lblOperation.Caption = "+"
End Sub

Private Sub cmdSubtract_Click()
    'Memory
    Dim AnsNumerator As Long, AnsDenom As Long
    Dim GCD As Long, LCM As Long
    'Input
    Call GetInput
    'Processing
    LCM = LeastCommonMultiple(Denom1, Denom2)
    AnsNumerator = LCM / Denom1 * Numerator1 - LCM / _
                                   Denom2 * Numerator2
    AnsDenom = LCM
    GCD = GreatestCommonDivisor(AnsNumerator, AnsDenom)
    'Output - Display answer as a reduced fraction
    txtAnsNumer.Text = (AnsNumerator / GCD)
    txtAnsDenom.Text = (AnsDenom / GCD)
    lblOperation.Caption = "-"
End Sub

Private Sub cmdDivide_Click()
    'Memory
    Dim AnsNumerator As Long, AnsDenom As Long
    Dim GCD As Long
    'Input
    Call GetInput
    'Processing
    AnsNumerator = Numerator1 * Denom2
    AnsDenom = Denom1 * Numerator2
    GCD = GreatestCommonDivisor(AnsNumerator, AnsDenom)
    'Output - Display answer as a reduced fraction
    txtAnsNumer.Text = AnsNumerator / GCD
    txtAnsDenom.Text = AnsDenom / GCD
    lblOperation.Caption = "/"
End Sub

Private Sub cmdMultiply_Click()
    'Memory
    Dim AnsNumerator As Long, AnsDenom As Long
    Dim GCD As Long
    'Input
    Call GetInput
    'Processing
    AnsNumerator = Numerator1 * Numerator2
    AnsDenom = Denom1 * Denom2
    GCD = GreatestCommonDivisor(AnsNumerator, AnsDenom)
    'Output - Display answer as a reduced fraction
    txtAnsNumer.Text = AnsNumerator / GCD
    txtAnsDenom.Text = AnsDenom / GCD
    lblOperation.Caption = "*"
End Sub

Private Sub GetInput()
    Denom1 = Val(txtDenom1.Text)
    Denom2 = Val(txtDenom2.Text)
    Numerator1 = Val(txtNumer1.Text)
    Numerator2 = Val(txtNumer2.Text)
End Sub

Private Function GreatestCommonDivisor(ByVal Num1 As _
                    Long, ByVal Num2 As Long) As Long
    Dim GCD As Long, Smaller As Long, _
                                PossibleDivisor As Long
    GCD = 1
    If Num1 < Num2 Then
        Smaller = Num1
    Else
        Smaller = Num2
    End If
    For PossibleDivisor = Smaller To 2 Step -1
        If Num1 Mod PossibleDivisor = 0 And _
                    Num2 Mod PossibleDivisor = 0 Then
            GCD = PossibleDivisor
            Exit For
        End If
    Next PossibleDivisor
    GreatestCommonDivisor = GCD
End Function

Private Function LeastCommonMultiple(ByVal Num1 As _
                    Long, ByVal Num2 As Long) As Long
    Dim PossibleMulti As Long, LCM As Long, _
                                        Smaller As Long
    If Num1 < Num2 Then
        Smaller = Num1
    Else
        Smaller = Num2
    End If
    'Find LCM
    For PossibleMulti = Smaller To Num1 * Num2
        If PossibleMulti Mod Num1 = 0 And _
                    PossibleMulti Mod Num2 = 0 Then
            LCM = PossibleMulti
            Exit For
        End If
    Next PossibleMulti
    LeastCommonMultiple = LCM
End Function
```

Using the Fraction Calculator Assignment to Learn How to Improve Existing Code (Part 3)

Instructions

Carefully study the code shown on the next page (Fraction Calculator Version 1.02). After you do so, run the "Fraction Calculator 1.02" VB program, which you will find stored in

I:\Out\Nolfi\Ics3m0\Fraction Calculator\ Fraction 1.02\Fraction 1.02.vbp

Then complete the following table.

How does version 1.02 differ from version 1.01?	
`Dim Denom1 As Long, Denom2 As Long` `Dim Numerator1 As Long, _` `Numerator2 As Long` In version 1.01, the above declarations had to be at the global (module) level. Why is it better to declare these variables locally in version 1.02?	
Use MSDN help or any other resources to do research on the following topics: • Arrays • Control Arrays	

```
'Fraction Calculator Version 1.02
'The code for this version is considerably shorter
'than the code for version 1.01. This is due to the
'elimination of a great deal of repetitive code in
'version 1.01.  Much of this was made possible
'by the use of control arrays.

Option Explicit
Dim Operation(0 To 3) As String * 1

Private Sub Form_Load()
    Operation(3) = "+"
    Operation(1) = "-"
    Operation(0) = "*"
    Operation(2) = "/"
End Sub

Private Sub cmdOperation_Click(Index As Integer)

    'Memory
    Dim Denom1 As Long, Denom2 As Long
    Dim Numerator1 As Long, Numerator2 As Long
    Dim AnsNumerator As Long, AnsDenom As Long
    Dim GCD As Long
    Const Add = 3, Subtract = 1, _
                            Multiply = 0, Divide = 2
    'Input
    Denom1 = Val(txtDenom1.Text)
    Denom2 = Val(txtDenom2.Text)
    Numerator1 = Val(txtNumer1.Text)
    Numerator2 = Val(txtNumer2.Text)
    'Processing
    If Index = Add Or Index = Subtract Then
        AnsNumerator = Numerator1 * Denom2 + _
                    (Index - 2) * Numerator2 * Denom1
        AnsDenom = Denom1 * Denom2
    ElseIf Index = Multiply Then
        AnsNumerator = Numerator1 * Numerator2
        AnsDenom = Denom1 * Denom2
    Else
        AnsNumerator = Numerator1 * Denom2
        AnsDenom = Denom1 * Numerator2
    End If

    GCD = GreatestCommonDivisor(AnsNumerator, AnsDenom)
    'Output - Display answer as a reduced fraction
    txtAnsNumer.Text = AnsNumerator / GCD
    txtAnsDenom.Text = AnsDenom / GCD
    lblOperation.Caption = Operation(Index)

End Sub

Private Function GreatestCommonDivisor(ByVal Num1 As _
                    Long, ByVal Num2 As Long) As Long

    Dim GCD As Long, Smaller As Long, _
                             PossibleDivisor As Long
    GCD = 1
    If Num1 < Num2 Then
        Smaller = Num1
    Else
        Smaller = Num2
    End If

    For PossibleDivisor = Smaller To 2 Step -1
        If Num1 Mod PossibleDivisor = 0 And _
                    Num2 Mod PossibleDivisor = 0 Then
            GCD = PossibleDivisor
            Exit For
        End If
    Next PossibleDivisor

    GreatestCommonDivisor = GCD

End Function
```

1. What does "`String * 1`" mean?
2. What is the purpose of this sub?
3. What is the purpose of multiplying by "`Index - 2`?"
4. What are these pieces of information called?
5. What are these variables called?
6. What is the purpose of this statement?

Function Procedures and Sub Procedures – Technical Information

Sub Procedures

Note: In the following formal descriptions of **Sub** procedures and **Function** procedures, the keywords enclosed in square brackets are optional and the "pipe" symbol (" | ") means "**OR**." For example, "[**Private** | **Public**]" means that either the keyword "**Private**" or the keyword "**Public**" ***may*** be used (but not both).

A **Sub** procedure is a block of code that is executed when ***invoked*** (called into action). By breaking the code in a module into **Sub** procedures, it becomes much easier to ***find***, ***modify*** and ***debug*** the code in your application. The syntax for a **Sub** procedure is:

[**Private**|**Public**] [**Static**] **Sub** *ProcedureName* (*FormalParameters*)

statements

End Sub

Each time the procedure is called, the ***statements*** between "**Sub**" and "**End Sub**" are executed. **Sub** procedures can be placed in standard modules, class modules and form modules. **Sub** procedures are by default **Public** in all modules, which means they can be called from anywhere in the application. The ***FormalParameters*** for a procedure are like a variable declaration, declaring values that are passed in from the calling procedure.

In Visual Basic, it is useful to distinguish between two types of **Sub** procedures, ***general procedures*** and ***event procedures***.

General Procedures

A ***general procedure*** tells the application how to perform a specific task. Once a general procedure is ***defined***, it must be specifically ***invoked*** (called into action) by the application. By contrast, an ***event procedure*** remains idle until called upon to respond to events caused by the user or triggered by the system.

Why create general procedures? One reason is that several different event procedures might need the same actions performed. A good programming strategy is to put common statements in a separate procedure (a general procedure) and have your event procedures call it. ***This eliminates the need to duplicate code and makes the application easier to maintain.***

Event Procedures

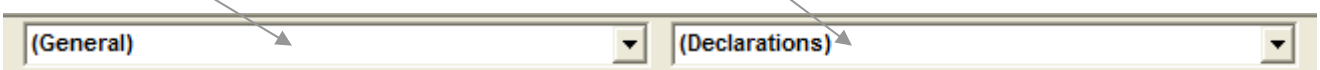
When an object in Visual Basic recognizes that an event has occurred, it automatically invokes the event procedure using the name corresponding to the event. Because the name establishes an association between the object and the code, event procedures are said to be ***attached*** to forms and controls.

- An event procedure for a control combines the **control's actual name** (specified in the Name property), an **underscore** (_), and the **event name**. For instance, if you want a command button named "**cmdPlay**" to invoke an event procedure when it is clicked, use the procedure "**cmdPlay_Click**."
- An event procedure for a form combines the word "**Form**," an **underscore** and the **event name**. If you want a form to invoke an event procedure when it is clicked, use the procedure "**Form_Click**." (Like controls, forms do have unique names, but they are not used in the names of event procedures.)

All event procedures use the same general syntax.

Syntax for a Control Event	**Syntax for a Form Event**
Private Sub *ControlName_EventName* (*Parameters*) *statements* **End Sub**	**Private Sub** Form_*EventName* (*Parameters*) *statements* **End Sub**

Although you can write event procedures from scratch, it is easier to use the code procedures provided by Visual Basic, which automatically include the correct procedure names. You can select a template in the **Code Editor** window by selecting an object from the **Object** box and then selecting a procedure from the **Procedure** box.



It is also a good idea to set the **Name** property of your controls before you start writing event procedures for them. If you change the name of a control after attaching a procedure to it, you must also change the name of the procedure to match the new name of the control. Otherwise, Visual Basic will not be able to match the control to the procedure. When a procedure name does not match a control name, it becomes a general procedure.

Function Procedures

Visual Basic includes built-in, or ***intrinsic*** functions, like Sqr, Cos or Chr. In addition, you can use the **Function** statement to write your own **Function** procedures.

The syntax for a **Function** procedure is:

[**Private**|**Public**] [**Static**] **Function** *ProcedureName* (*FormalParameters*) [**As** *type*]

statements

End Function

Like a Sub procedure, a Function procedure is a separate procedure that can take parameters, perform a series of statements and change the value of its parameters. Unlike a **Sub** procedure, a **Function** procedure can return a value to the calling procedure. There are several differences between **Sub** and **Function** procedures:

- Generally, you call a function by including the function procedure name and arguments on the right side of a larger statement or expression (*returnvalue = function( )*).
- Function procedures have data types, just as variables do. This determines the type of the return value. (In the absence of an "**As**" clause, the type is the default **Variant** type.)
- You return a value by assigning it to the *ProcedureName* itself. When the **Function** procedure returns a value, this value can then become part of a larger expression.
- Although a **Function** procedure is allowed to alter the values of the *Arguments* in the call to the function, to allow a function to do so is generally considered poor programming style. In most cases, a **Function** procedure should simply return a value without altering the values of any variables other than its own local variables. In addition, a **Function** should not trigger any input or output operations. **Function** procedures that alter variables or that trigger I/O (input/output) operations are said to have ***side effects***. **Do not write Function procedures that have side effects! Use Sub procedures instead!**
- **Sub** procedures are used when it is necessary for a procedure to complete *several tasks*. **Function** procedures are used when the only task required is to compute a *single value*.

Examples Including Terminology

```
'The following function procedure returns the distance between two points
Private Function DistBetweenPoints (ByVal X1 As Double, ByVal Y1 As Double, ByVal X2 _
                                    As Double, ByVal Y2 As Double) As Double

   DistBetweenPoints = Sqr( (X2 – X1)^2 + (Y2 – Y1)^2 )

End Function
```

The keyword "**ByVal**" means that the parameters are passed "by value." Parameters declared using "**ByVal**" store ***copies*** of the values passed in the ***call*** of the function. This protects any variables in the call from being altered accidentally.

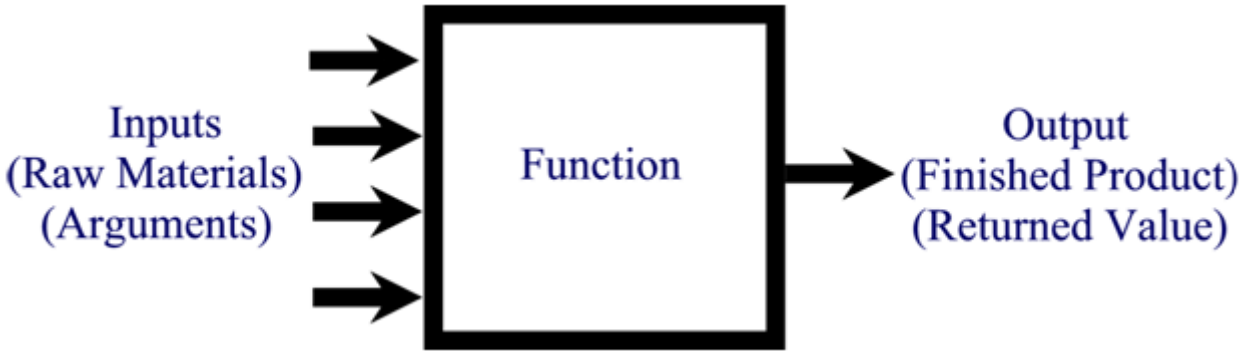
The variables in this list are called the ***formal parameters*** or simply the ***parameters*** of the function.

This is called the ***definition*** of the function procedure.

The values in the call that are passed to the *formal parameters* in the function definition are called the ***actual parameters*** or ***arguments*** of the function.

```
' The following is an example of a call to the above function.  The distance between the
' points (1, 4) and (7.8, 9.9) is returned and assigned to the variable "Dist"
Dim Dist As Double
Dist = DistBetweenPoints (1, 4, 7.8, 9.9)
```

A Function is like a Machine



NOTE: Functions in programming are based on the concept of a mathematical function. For instance, when we write $f(x) = x^2$ we mean that the "input" to the function is x and the output is x^2. Although you probably have not encountered any thus far in your education, it is possible to define functions of more than one variable. For instance, the function $f : \mathbb{R}^2 \to \mathbb{R}$ defined by $f(x, y) = x^2 + y^2$ has two inputs x and y (which are both real numbers) and one output (which is also a real number).

Exercises

1. Write a function that takes two integer parameters ("Lowest" and "Highest") and returns a pseudo-random integer greater than or equal to "Lowest" and less than or equal to "Highest."

2. Write a function that returns the length of the hypotenuse of a right triangle given the lengths of the other two sides.

3. Try to write a function that calculates and returns the midpoint of a line segment. What difficulties do you encounter while trying to write this function? See **I:\Out\Nolfi\Ics3m0\Midpoint and Length** for a solution to this problem.

4. What are the differences between ***general sub procedures*** and ***event sub procedures***? How does Visual Basic detect whether you are creating a general sub procedure or an event sub procedure?

5. What are the main differences between sub procedures and function procedures? Under what circumstances should you use a function procedure and under what circumstances should you use a sub procedure? Provide specific examples.

6. What are side effects and why should we always avoid writing functions that have side effects? If you write a function procedure that has side effects, why should you consider rewriting it as a sub procedure?

7. Explain the difference between ***defining*** a procedure and ***calling*** a procedure.

8. Explain the difference between an ***intrinsic*** function and a ***programmer-defined*** function. While developing a piece of software, how would you decide whether you need to create a procedure (sub or function) to complete a certain task?

9. Explain the difference between ***formal parameters*** (parameters) and ***actual parameters*** (arguments).

10. Explain the difference between declaring a procedure to be **Private** and declaring a procedure to be **Public.** If you neglect to specify **Private** or **Public,** which will Visual Basic use by default?

11. What is the purpose of the **ByVal** keyword in the formal parameter list of a VB procedure? Why is it wise to use the **ByVal** keyword for function procedure formal parameters?

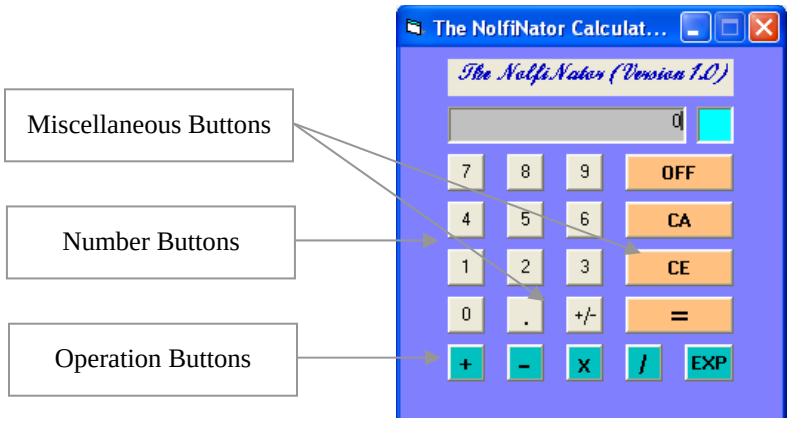
12. Explain the differences between a control event and a form event.

13. Define "syntax" and "parameter."

Examples Showing the Differences between Function and Sub Procedures

An Example of a Sub Procedure

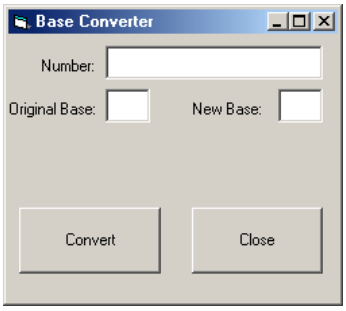
```
'Enable or disable groups of buttons
Private Sub EnableOrDisableButtonGroups(ByVal Enable As Boolean, ByVal Numbers As Boolean, _
                                        ByVal Operations As Boolean, ByVal Others As Boolean)
    Dim I As Byte
    If Numbers = True Then
        For I = 0 To 9
            cmdNumber(I).Enabled = Enable
        Next I
    End If
    If Operations = True Then
        For I = 0 To 4
            cmdOperation(I).Enabled = Enable
        Next I
    End If
    If Others = True Then
        cmdDecimalPt.Enabled = Enable
        cmdPlusMinus.Enabled = Enable
        cmdCE.Enabled = Enable
        cmdEXP.Enabled = Enable
    End If
End Sub
'An example of a call to the above sub procedure
Call EnableOrDisableButtonGroups(True, False, False, True)
```



Take note of the following features of the sub procedure shown above:

- several different tasks are being performed, including many that affect several ***global objects***
- the procedure is written in a ***general manner*** (i.e. it can complete a variety of different tasks depending on the values of the parameters)
- the "**ByVal**" keyword used in the declarations of the formal parameters is used to prevent this sub procedure from changing the values of variables in calls to the procedure
- Subs ***do not*** return a value

An Example of a Function Procedure




```
Option Explicit
Private Sub cmdConvert_Click()
    lblNewNumber.Caption = "Number in new base: " & _
                           ChangeBase(txtNumber.Text, _
                           Val(txtOldBase.Text), Val(txtNewBase.Text))
    lblNewNumber.Visible = True
End Sub
Private Sub cmdClose_Click()
    Unload frmBaseConverter
    End
End Sub
Private Sub txtNumber_Change()
    txtOldBase.Text = ""
    txtNewBase.Text = ""
    lblNewNumber.Visible = False
End Sub
'"Number" is converted from "OldBase" to "NewBase"
Private Function ChangeBase(ByVal Number As String, ByVal OldBase As Byte, ByVal NewBase As Byte) As String
    Dim Remainder As Byte, Quotient As Byte, Pos As Byte, LenNum As Byte
    Dim BaseTen As Double, NewNumber As String
    Const Digits = "0123456789ABCDEF"
    'Convert "Number" expressed in "OldBase" to base 10
    BaseTen = 0
    LenNum = Len(Number)
    For Pos = 1 To LenNum
        BaseTen = BaseTen + (InStr(Digits, UCase(Mid(Number, Pos, 1))) - 1) * OldBase ^ (LenNum - Pos)
    Next Pos
    'Convert "BaseTen" to "NewBase"
    NewNumber = ""
    Quotient = BaseTen
    Do
       Remainder = Quotient Mod NewBase
       Quotient = Int(Quotient / NewBase)
       NewNumber = Mid(Digits, Remainder + 1, 1) & NewNumber
    Loop Until Quotient = 0
    ChangeBase = NewNumber
End Function
```

Take note of the following features of "ChangeBase"

- it does not directly access or alter any global variables or objects
- it receives all required data through its parameters
- all the code within the function exists for the ultimate purpose of returning a single value
- a single **String** value ***is returned*** according to the value of 'ChangeBase' when the function halts its execution

Review of Unit 2

Critically Important Problem Solving Strategies for Programming

1. Solve a specific example of the problem
2. Plan your solution in a logical, organized fashion
3. Break up a large complex problem into several smaller, simpler problems

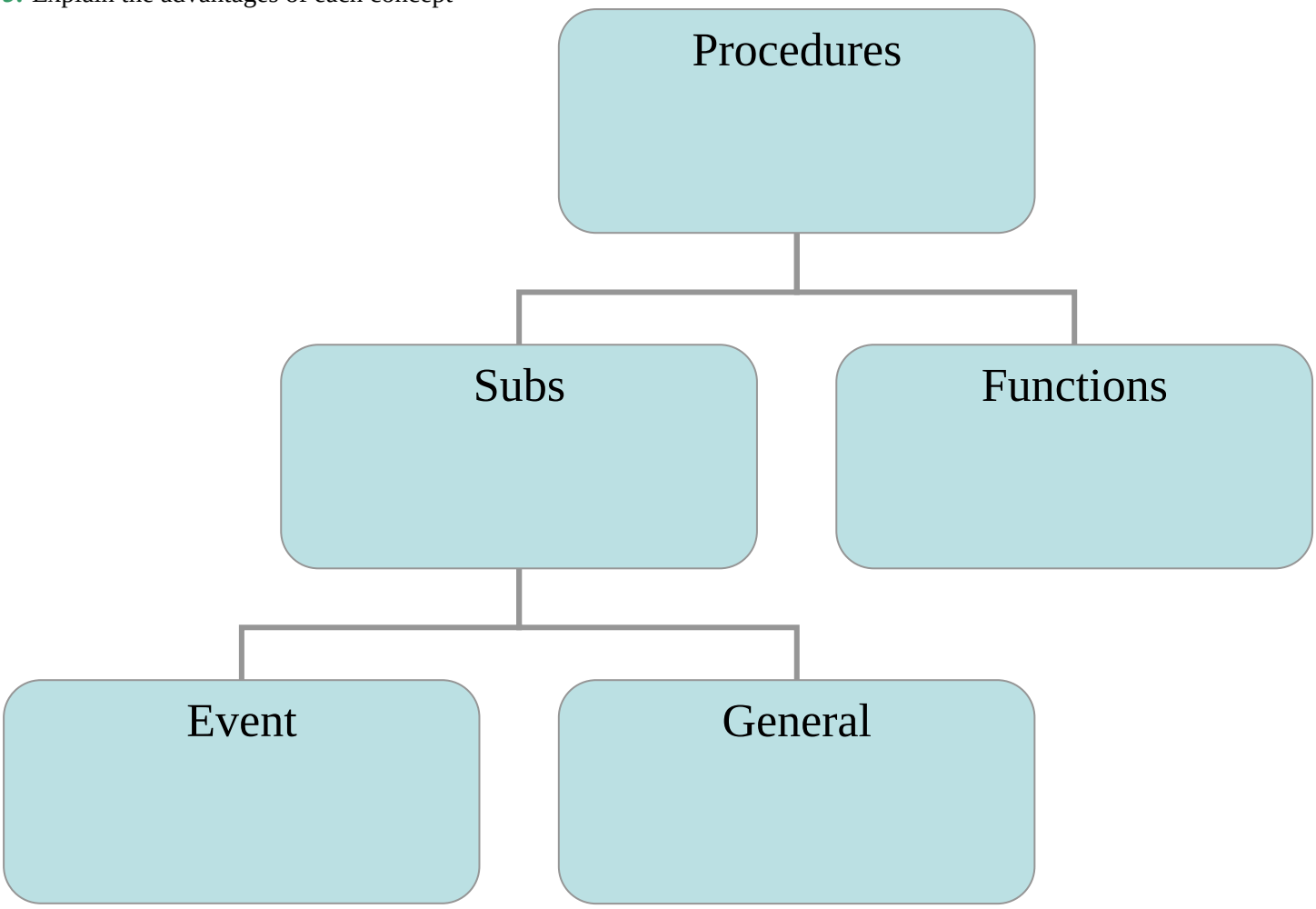
Additional General Problem Solving Strategies

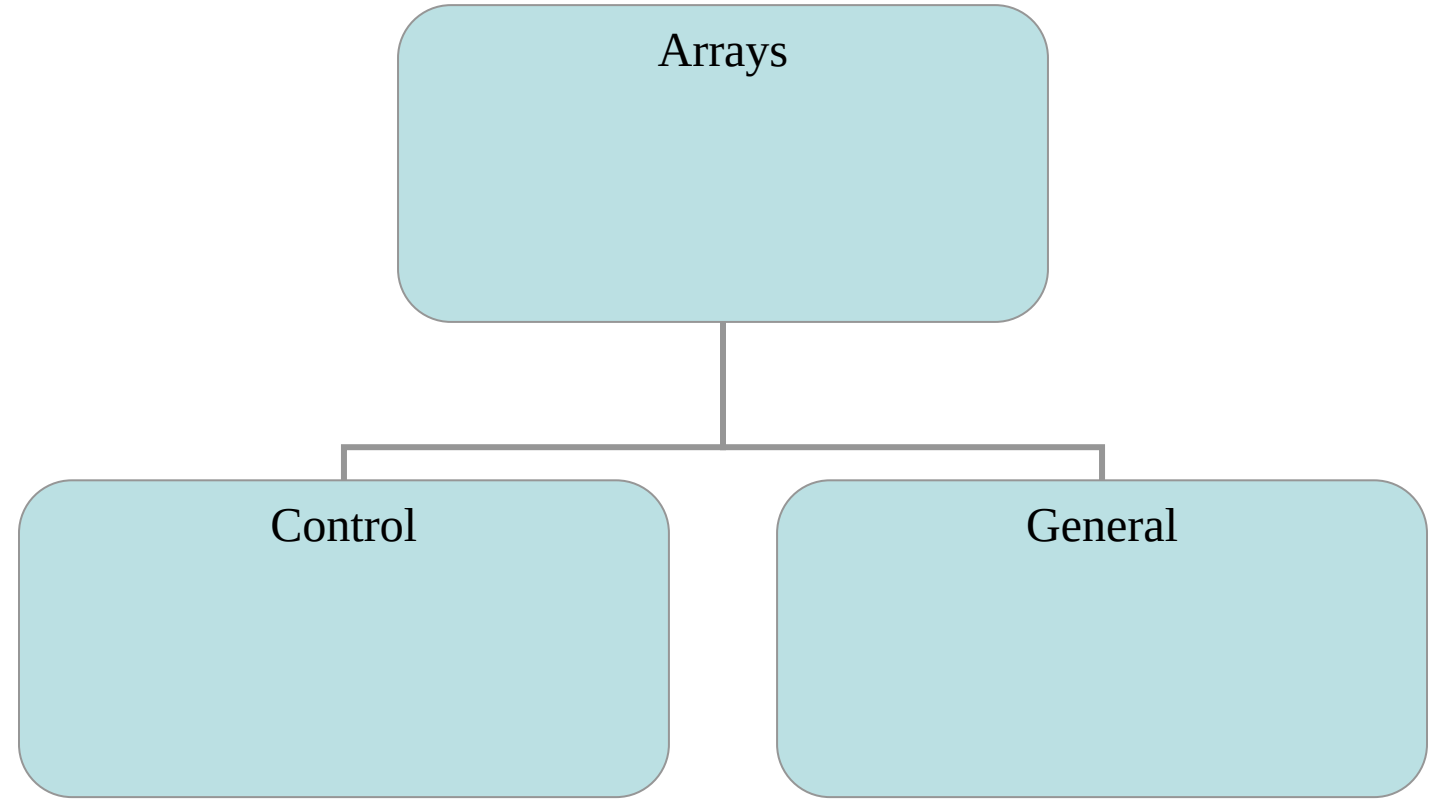
1. Solve a simpler but related problem
2. Make reasonable, simplifying assumptions
3. Look for patterns
4. Draw diagrams
5. Do research to find out if anyone else has solved the problem

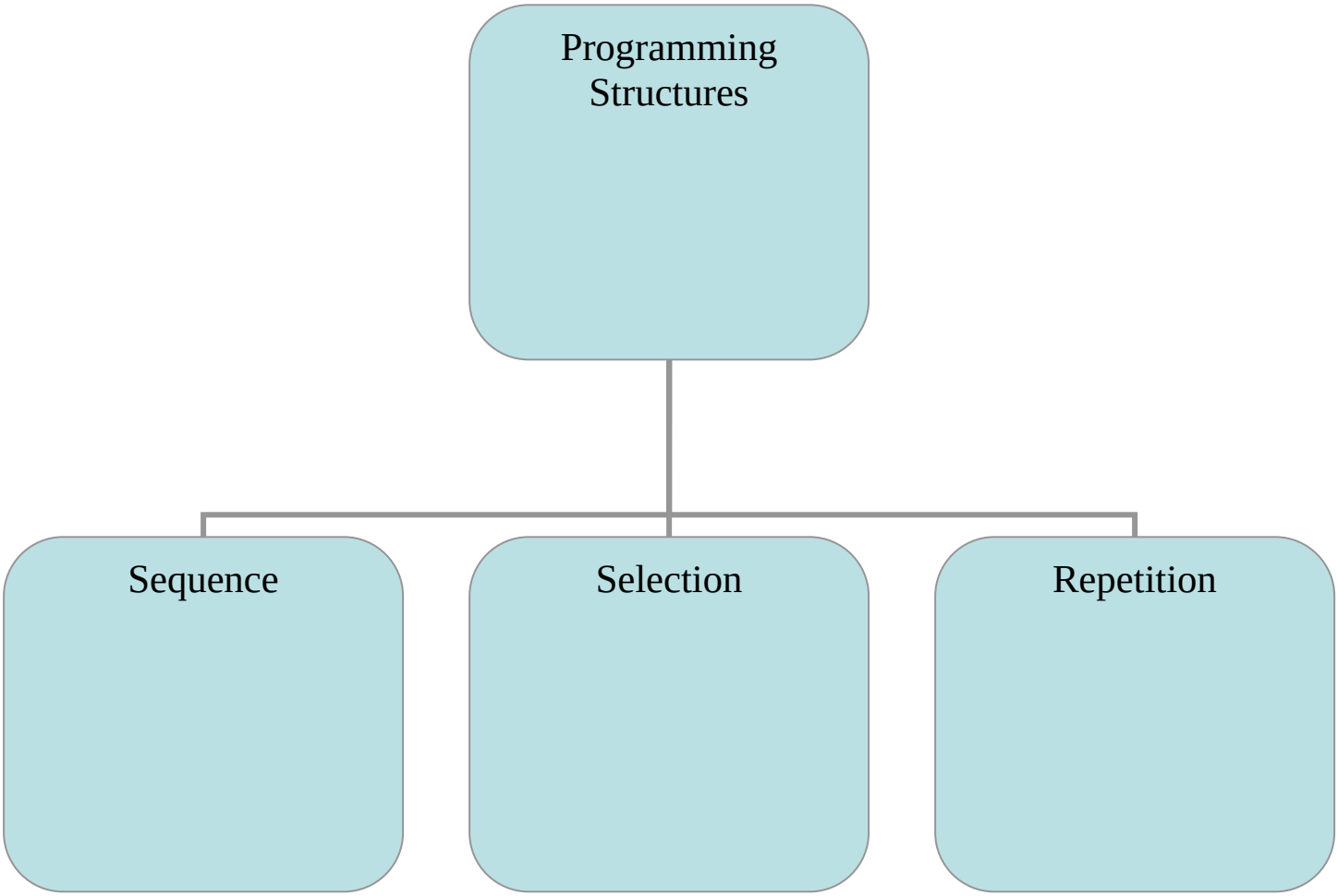
Important Programming Concepts

Complete each of the following diagrams.

1. Explain each concept
2. Explain the uses of each concept
3. Explain the advantages of each concept




Arrays
Control
General


Programming Structures
Sequence
Selection
Repetition

Generating Pseudo-Random Integers

To generate pseudo-random integers greater than or equal to "`Lowest`" and less than or equal to "`Highest`" use the VB expression

```
Int(Rnd*(Highest–Lowest+1)+Lowest)
```

For example, the expression `Int(Rnd*6+1)` is used to generate a pseudo-random integer from 1 to 6 inclusive.

Integer Division and Remainder

\ → to obtain ***quotient*** of division of two integers **Mod** → to obtain ***remainder*** of division of two integers

These operators were very useful in the "Time Converter" and "Coins and Bills" problems. In the grade 12 computer science course, you will discover that these operations (/ and % respectively in C, C++, Java) are very useful in the "Roman Converter" problem (convert between Arabic and Roman forms).

Sequence, Selection and Repetition

These are the ***main structures*** in programming. Any program that can be written will use some combination of these three structures.

"If" Statements

Structure to use when exactly ONE Group of Statements is to be Selected and all others Rejected		*Structure to use when the Conditions are Independent of each other*	
If *condition1* **Then** *groupOfStatements1* **ElseIf** *condition2* **Then** *groupOfStatements2* **ElseIf** *condition3* **Then** *groupOfStatements3* . . . **Else** *groupOfStatementsN* **End If**	Only ONE of these groups of statements is executed. As evaluated from top to bottom, if *conditionM* is the first condition found to be true, then *groupOfStatementsM* is executed and all others are rejected. If *conditionM* is false for all values of *M*, then *groupOfStatementsN* (in the "**Else**" clause) is executed.	**If** *condition1* **Then** *groupOfStatements1* **End If** **If** *condition2* **Then** *groupOfStatements2* **End If** **If** *condition3* **Then** *groupOfStatements3* **End If** . . .	The structure shown in this case should be used whenever the conditions are unrelated to one another. For instance, whether *condition1* is true has ***nothing to do with*** whether *condition2* and *condition3* are true.

Data Types and Encoding Schemes

Integers (Whole Numbers)			*Floating Point Numbers*			*Text*			*Logical Values*		
VB Type	*Operations*	*Encod. Scheme*	*VB Type*	*Operations*	*Encod. Scheme*	*VB Type*	*Operations*	*Encod. Scheme*	*VB Type*	*Operations*	*Encod. Scheme*
Byte	+, –, *, /, \, ^, **Mod**	8-bit unsigned integer (binary)	**Single**	+, –, *, /, ^	IEEE754 32-bit	**String**	&	Unicode	**Boolean**	**And**, **Or**, **Not**	16-bit unsigned integer
Integer	+, –, *, /, \, ^, **Mod**	16-bit signed integer (twos complement binary)	**Double**	+, –, *, /, ^	IEEE754 64-bit						
Long	+, –, *, /, \, ^, **Mod**	32-bit signed integer (twos complement binary)									

Some Useful Intrinsic (Built-In) Functions

State the purpose of each of the following intrinsic functions.

Val _____

CStr _____

Trim _____

Format _____

Sqr _____

Important Terminology

Explain each of the following terms. In addition, provide an example of each. (The first one is done for you.)

Term	*Explanation*	*Example*
Assignment Statement	A statement in which a value is assigned (given) to a variable.	RipoffGameConsole = "PS3"
Expression		
Operator		
Keyword		
Data Type		
Object		
Event		
Property		
Method		
Procedure		
Sub Procedure		
Function Procedure		
Statement Continuation Character		
Compound Condition		
Variable Declaration		
DoEvents		
Global Variables		
Local Variables		
Parameters		